UNLocBoX: Documentation

MATLAB convex optimization toolbox

Lausanne - November 2012

Perraudin Nathanaël, Shuman David
Vandergheynst Pierre and **Puy Gilles**

LTS2 - EPFL

Contents

Chapter 1

Unlocbox - Solvers

1.1 Universal solver

1.1.1 SOLVEP - solve a minimization problem

Usage

```
sol = solvep(x_0, F, param);
sol = solvep(x_0, F);
[sol,infos,objectiv] = solvep(...);
```

Input parameters

x_0	Starting point of the algorithm
F	array of function to minimize (structure)
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`solvep` solves:

$$sol = arg\min_x |sum_i f_i(x) \qquad for \qquad x \in R^N$$

where x is the variable.

x_0 is the starting point of the algorithm. A good starting point could significantly reduce the computation time

F is an array of structure representing convex function to be minimized. These functions can be minimized thanks to: 1) their gradient (only if they are differentiable) OR 2) their proximal operator. As a result the algorithm will need at least one of the above. To define a function *f1* you usually need to either create a structure with the fields 1) f1.eval AND 2) f1.prox that is needed in case of non-differentiable functions *f1*, OR a structure with the fields 1) f1.eval AND 2) f1.grad AND 3) f1.beta The fields f1.eval, f1.prox and f1.grad contain an inline function that computes respectively the evaluation of the function *f1* itself, its proximal operator or its gradient.

The field f1.beta usually needed for differentiable functions is an upper bound of the Lipschitz constant of the gradient of f1 (i.e. the squared norm of the gradient operator).

Depending on the solver, not all this operators are necessary. Also, depending on the existence of the above field, solvep chooses a different solver. See each solver documentation for details.

When three functions are defined, F = {f1, f2, f3}, then primal dual algorithms are used, in that case the linear operator that brings us from the primal to the dual space and the adjoint operator should be defined: 1) `f1.L` : linear operator, matrix or operator (default identity) 2) `f1.Lt` : adjoint of f1.L, matrix or operator (default identity) 3) `f1.norm_L` : upper bound of the norm of operator L (default: 1)

param a Matlab structure containing the following fields:

- *param.gamma* : is the step size. Watch out, this parameter is bounded. It should be below $1/\beta$ (*f2* is β Lipchitz continuous). By default, it is computed with the lipschitz constant of all smooth functions.
- *param.tol* : Tolerance to stop iterating. Please see *param.stopping_criterion*. (Default 1e-4).
- *param.algo* : solver used for the problem. Determined automatically with the functions in *f*.
- *param.stopping_criterion* : is stopping criterion to end the algorithm. Possible values are:
  - 'rel_norm_obj' : Relative norm of the objective function.
  - 'rel_norm_primal' : Relative norm of the primal variables.
  - 'rel_norm_dual' : Relative norm of the dual variables.
  - 'rel_norm_primal_dual' : Relative norm of the primal and the dual variables.
  - 'obj_increase' : Stops when the objective function starts increasing or stay equal.
  - 'obj_threshold' : Stops when the objective function is below a threshold. The threshold is set in *param.tol*.

  For the 'rel_norm' stopping criterion, the algorithm end if

$$\frac{\|n(t)-n(t-1)\|_2}{\|n(t)\|_2} < tol,$$

  where $n(t)$ is the objective function, the primal or the dual variable at iteration t.
- *param.maxit* : is the maximum number of iteration. By default, it is 200.
- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps.
- *param.debug_mode* : Compute all internal convergence parameters. Activate this option for debugging

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.crit* : Stopping critterion used

Additionally, depending on the stopping critterion, the structure *info* also contains:

- *info.objective* : Value of the objective function
- *info.rel_norm_obj* : Relative norm of the objective function.

- *info.rel_norm_primal* : Relative norm of the primal variable.

If the flag *param.debug_mode* is activated, the previous quantity are always computed. Moreover, for solver using dual variable, *info* also contains:

- *info.rel_norm_dual* : Relative norm of the dual variable.
- *info.dual_var* : Final dual variables.

1.2 General solvers

1.2.1 ADMM - alternating-direction method of multipliers

Usage

```
sol = admm(x_0,f1,f2,param);
sol = admm(x_0,f1,f2);
[sol,info,objective] = admm(...);
```

Input parameters

x_0	Starting point of the algorithm
f1	First function to minimize
f2	Second function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`admm` (using alternating-direction method of multipliers) solves:

$$sol = \min_x f_1(y) + f_2(x) \qquad s.t. \qquad y = Lx$$

where x is the optimization variable.

Please read the paper of Boyd "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers" to be able to understand this demonstration file.

f1 is a structure representing a convex function. Inside the structure, there have to be the prox of the function that can be called by *f1.proxL* and the function itself that can be called by *f1.eval*. WARNING !!! The proxL of *f1* is not the usual prox! But the solution to this problem:

$$prox^L_{f_1,\gamma}(z) = \min_x \frac{1}{2}\|Lx - z\|_2^2 + \gamma f_1(x)$$

f2 is a structure representing a convex function. Inside the structure, there have to be the prox of the function that can be called by *f2.prox* and the function itself that can be called by *f2.eval*. The prox of *f2* is the usual prox:

$$prox_{f_2,\gamma}(z) = \min_x \frac{1}{2}\|x-z\|_2^2 + \gamma f_2(x)$$

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.L* : linear operator that link x and y: $y = Lx$. This operator can be given in a matrix form (default Identity) or as a function handle.

References: [1], [2]

1.2.2 DOUGLAS_RACHFORD - Douglas-rachford proximal splitting algorithm

Usage

```
sol = douglas_rachford(x_0,f1, f2, param);
sol = douglas_rachford(x_0,f1, f2);
[sol, info] = douglas_rachford(...);
```

Input parameters

x_0	Starting point of the algorithm
f1	First function to minimize
f2	Second function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`douglas_rachford` algorithm solves:

$$sol = arg\min_x f_1(x) + f_2(x) \qquad for \qquad x \in R^N$$

where x is the variable.

- *f1* and *f2* are structures representing convex functions. Inside the structure, there have to be the prox of the function that can be called by *f1.prox* and the function itself that can be called by *f1.eval*.

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.lambda*: is the weight of the update term. It is kind of a timestep for the proximal operators. (Warning it should not be confused with *gamma*, the time step for gradient descent part). By default it is set to 1. Do not change this parameter unless you know what you do.

References: [3]

1.2.3 FORWARD_BACKWARD - Forward-backward splitting algorithm

Usage

```
sol = forward_backward(x_0,f1, f2, param);
sol = forward_backward(x_0,f1, f2);
[sol,infos] = forward_backward(...);
```

Input parameters

x_0	Starting point of the algorithm
f1	First function to minimize
f2	Second function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`forward_backward` solves:

$$sol = arg\min_x f_1(x) + f_2(x) \qquad for \qquad x \in R^N$$

where x is the optimization variable.

f1 is a structure representing a convex function. Inside the structure, there have to be the prox of the function that can be called by *f1.prox* and the function itself that can be called by *f1.eval*.

f2 is a structure representing a convex function, with a β Lipschitz continuous gradient. Inside the structure, there have to be the gradient of the function that can be called by *f2.grad* and the function itself that can be called by *f2.eval*.

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.lambda* : is the weight of the update term. It is kind of a timestep for the proximal operators. (Warning it should not be confused with *gamma*, the time step for gradient descent part). By default it is set to 1. Do not change this parameter unless you know what you do.
- *param.method* : is the method used to solve the problem. It can be the fast version 'FISTA' or 'ISTA'. By default, it's 'FISTA'.

References: [4], [3]

1.2.4 GENERALIZED_FORWARD_BACKWARD - Generalized forward backward algorithm

Usage

```
sol = generalized_forward_backward(x_0,F, f2, param);
sol = generalized_forward_backward(x_0,F, f2);
[sol, info] = generalized_forward_backward(...);
```

Input parameters

x_0	Starting point of the algorithm
F	Array of structure representing the functions to minimize
f2	Another function to minimize with a known gradient
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`generalized_forward_backward` solves:

$$sol = \min_z f_2(z) + \sum_i w_i F_i(z) \qquad for \qquad z \in R^N$$

With z the variable and *wi* the weight accorded to every term of the sum

- *x_0* : is the starting point.
- *F* is a cellarray of structures representing functions containing operators inside and eventually the norm. The prox: *F{i}.prox* and the function: *F{i}.eval* are defined in the same way as in the Forward-backward and Douglas-Rachford algorithms
- *f2* is a structure representing a convex function, with a beta Lipschitz continuous gradient. Inside the structure, there have to be the gradient of the function that can be called by *f2.grad* and the function itself that can be called by *f2.eval*.
- *param* is a Matlab structure containing the following fields:
  - ***param.weights*** (**weights of different functions (default =** $1/N$**,**) where N is the total number of function)
  - *param.lambda*: is the weight of the update term. By default 1. This should be between 0 and 1.

References: [5]

1.2.5 PPXA - Parallel Proximal algorithm

Usage

```
sol = ppxa(x_0, F, param);
sol = ppxa(x_0, F);
[sol, infos] = ppxa(...);
```

Input parameters

x_0	Starting point of the algorithm
F	Array of function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`ppxa`, derived from the Douglas-Rachford algorithm, solves

$$sol = \min_x \sum_i W_i f_i(x)$$

for x in R^N, where x is the variable and *x_0* is the starting point.

F is a cellarray of structures representing functions. All of them should contains at least two fields. *F{ii}.eval* to evaluate the function and *F{ii}.prox* to compute the proximal operator of the function.

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.W* : the weight (all equal by default)
- *param.lambda*: is the weight of the update term. It is kind of a timestep for the proximal operators. (Warning it should not be confused with *gamma*, the time step for gradient descent part). By default it is set to 0.99. Do not change this parameter unless you know what you do.

References: [2]

1.2.6 SDMM - Simultaneous-direction method of multipliers algorithm

Usage

```
sol = sdmm(F,param);
sol = sdmm(F);
[sol,info] = sdmm(...);
```

Input parameters

F	Array of function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`sdmm`, from simultaneous-direction method of multipliers solves:

$$sol = \min_x \sum_i f_i(L_i x)$$

where x belong to R^N, L_i are linear operators and x_i are the minimization variables.

F is a cellarray of structure representing the functions. In the function `F{i}`, there have to be:

- `F{i}.eval(x_i)` : an operator to evaluate the function

- `F{i}.prox(x_i, gamma)` : an operator to evaluate the prox of the function
- `F{i}.x0` : vector of initial value

Optionally you can define

- `F{i}.L` : linear operator, matrix or operator (default identity)
- `F{i}.Lt` : adjoint of linear operator, matrix or operator (default identity)

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\max_i \frac{\|y_i(t) - y_i(t-1)\|}{\|y_i(t)\|} < tol,$$

where $y_i(t)$ are the dual variable of function i at itertion t by default, `tol=10e-4`.

Warning! This stopping criterion is different from other solver!

- *param.Qinv* : Inverted Q matrix. $Qinv = Q^{-1}$ with:

$$Q = \sum_i L_i^T (L_i x)$$

By default, Qinv is the identity matrix divided by the number of functions.

This parameter can be given in a matrix form or in a linear operator form.

References: [3], [2]

1.2.7 FB_BASED_PRIMAL_DUAL - forward backward based primal dual

Usage

```
sol = fb_based_primal_dual(x_0,f1,f2, f3,param);
sol = fb_based_primal_dual(x_0,f1,f2,f3);
[sol,info] = fb_based_primal_dual(...);
```

Input parameters

x_0	Starting point of the algorithm
f1	First function to minimize
f2	Second function to minimize
f3	Third function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`fb_based_primal_dual` solves:

$$sol = \min_x f_1(x) + f_2(Lx) + f_3(x)$$

where x is the optimization variable with f_1 or f_3 a smooth function and L a linear operator. f_1 and f_3 are defined like other traditional functions.

Note that *f2* is a structure of a functions with:

- `f2.eval(x_i)` : an operator to evaluate the function
- `f2.prox(x_i, gamma)` : an operator to evaluate the prox of the function

Optionally you can define

- `f2.L` : linear operator, matrix or operator (default identity)
- `f2.Lt` : adjoint of linear operator, matrix or operator (default identity)
- `f2.norm_L` : bound on the norm of the operator L (default: 1), i.e.

$$\|Lx\|^2 \leqslant \nu \|x\|^2$$

The default choice for the time-step makes the following

$$\frac{1}{\tau} - \sigma\nu = \frac{\beta}{2}$$

with additionnaly

$$\frac{1}{2\tau} = \sigma\nu = \frac{\beta}{2}$$

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\max_i \frac{\|y_i(t) - y_i(t-1)\|}{\|y_i(t)\|} < tol,$$

  where $y_i(t)$ are the dual variable of function i at itertion t by default, `tol=10e-4`.

  Warning! This stopping criterion is different from other solver!

- *param.tau* : first timestep.
- *param.sigma* : second timestep. The timesteps should satisfy the following relationship (beta is the lipschitz constant of the smooth term):

$$\frac{1}{\tau} - \sigma\nu \geqslant \frac{\beta}{2}$$

- *param.rescale* : Use the rescaled version of the algorithm (default 0)
- *param.method* : is the method used to solve the problem. It can be the fast version 'FISTA' or 'ISTA'. By default, it's 'ISTA'.

References: [6]

1.2.8 FBF_PRIMAL_DUAL - forward backward forward primal dual

Usage

```
sol = fbf_primal_dual(x_0,f1, f2, f3, param);
sol = fbf_primal_dual(x_0,f1, f2, f3);
[sol,info,objective] = fbf_primal_dual(...);
```

Input parameters

x_0	Starting point of the algorithm
f1	First function to minimize
f2	Second function to minimize
f3	Third function to minimize
param	Optional parameters

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`fbf_primal_dual` (using forward backward forward based primal dual) solves:

$$sol = \min_x f_1(x) + f_2(Lx) + f_3(x)$$

where x is the optimization variable with f_1 or f_3 a smooth function and L a linear operator. f_1 and f_3 are defined like other traditional functions.

Note that *f2* is a structure of a functions with:

- `f2.eval(x_i)` : an operator to evaluate the function
- `f2.prox(x_i, gamma)` : an operator to evaluate the prox of the function

Optionally you can define

- `f2.L` : linear operator, matrix or operator (default identity)
- `f2.Lt` : adjoint of linear operator, matrix or operator (default identity)
- `f2.norm_L` : bound on the norm of the operator L (default: 1), i.e.

$$\|Lx\|^2 \leqslant \nu \|x\|^2$$

param a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:

- *param.tol* : is stopping criterion for the loop. The algorithm stops if

$$\max_i \frac{\|y_i(t) - y_i(t-1)\|}{\|y_i(t)\|} < tol,$$

where $y_i(t)$ are the dual variable of function i at itertion t by default, `tol=10e-4`.

Warning! This stopping criterion is different from other solvers!

- *param.mu* : parameter mu of paper [1]
- *param.epsilon*: parameter epsilon of paper [1]
- *param.normalized_timestep*: from 0 to 1, mapping to [epsilon, (1-epsilon)/mu]

References: [6]

1.2.9 GRADIENT_DESCENT - Gradient descent using the forward backward algorithm

Usage

```
sol = gradient_descent(x_0,F, param);
sol = gradient_descent(x_0,F);
[sol,info] = gradient_descent(...);
```

Input parameters

x_0	Starting point of the algorithm
F	Functions to be minimized
param	Optional parameter

Output parameters

sol	Solution
info	Cell array of functions

Description

`gradient_descent` solves:

$$sol = arg\min_x \sum_i f_1(x) \qquad for \qquad x \in R^N$$

where x are the optimization variables.

F is a cell array of structure object. Each structure represent one function to be minimized. They all contains a field *F{ii}.eval* that is a implicite function to evaluate the corresponding function and a field *F{ii}.grad* that is another implicite function to compute the gradient of the function. Please, specify also, the Lipschitz constant of the gradient in *F{ii}.beta*.

1.2.10 POCS - Projection onto convex sets

Usage

```
sol = pocs(x_0,F, param);
sol = pocs(x_0,F);
[sol,info] = pocs(...);
```

Input parameters

x_0	Starting point of the algorithm
F	Array of function to minimize
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`pocs` solves:

$$sol = arg\min_x \|x - x_0\| \qquad for \qquad x \in \cap_i \mathscr{C}_i$$

where x are the optimization variables.

F is a cell array of structures representing the indicative functions of all sets. F{ii}.eval contains an anonymous function that evaluate how far is the point x to the set ii. F{ii}.prox project the point x to the set ii. This .prox notation is kept for compatibility reason.

This function is kept for backward compatibility and is not recommended to be used.

1.3 Composed solvers

1.3.1 RLR - Regularized Linear Regression

Usage

```
sol = rlr(x_0,f,A,At, param)
sol = rlr(x_0,f,A,At)
[sol, info] = rlr(..,)
```

Input parameters

x_0	Starting point of the algorithm
f	Function to minimize
A	Operator
At	Adjoint operator
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

This function solve minimization problem using forward-backward splitting
`sol = rlr(x_0,f,A,At, param)` solves:

$$sol = arg\min_x \|x_0 - Ax\|_2^2 + f(x) \qquad for \qquad x \in R^N$$

where x is the variable.

- *x_0* is the starting point.
- f is a structure representing a convex function. Inside the structure, there have to be the prox of the function that can be called by *f.prox* and the function itself that can be called by *f.eval*.
- A is the operator
- *At* is the adjoint operator of A
- *param* a Matlab structure containing solver paremeters. See the function solvep for more information. Additionally it contains those aditional fields:
  - *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

  - *param.method* : is the method used to solve the problem. It can be 'FISTA' or 'ISTA'. By default, it's 'FISTA'.

References: [3]

1.3.2 SOLVE_BPDN - Solve BPDN (basis pursuit denoising) problem

Usage

```
sol = solve_bpdn(y, epsilon, A, At, Psi, Psit, param)
sol = solve_bpdn(y, epsilon, A, At, Psi, Psit)
[sol, info] = solve_bpdn(...)
```

Input parameters

y	Measurements
epsilon	Radius of the L2 ball
A	Operator
At	Adjoint of A
Psi	Operator
Psit	Adjoint of Psi
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`sol = solve_BPDN(y, A, At, Psi, Psit, param)` solves:

$$arg\min_x \|\Psi x\|_1 s.t. \|y - Ax\|_2 < \varepsilon$$

Y contains the measurements. A is the forward measurement operator and At the associated adjoint operator. Psit is a sparfying transform and Psi its adjoint. PARAM a Matlab structure containing the following fields:

General parameters:

- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t) - n(t-1)}{n(t)} < tol,$$

  where $n(t) = ||\Psi(x)||$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.gamma* : control the converge speed (default: 1).

Projection onto the L2-ball :

- *param.tight_b2* : 1 if A is a tight frame or 0 if not (default = 1)
- *nu_b2* : bound on the norm of the operator A, i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

- *tol_b2* : tolerance for the projection onto the L2 ball (default: 1e-3):

$$\frac{\varepsilon}{1 - tol} \leqslant \|y - Az\|_2 \leqslant \frac{\varepsilon}{1 + tol}$$

- *maxit_b2* : max. nb. of iterations for the projection onto the L2 ball (default 200).

Proximal L1 operator:

- *tol_l1* : Used as stopping criterion for the proximal L1 operator. Min. relative change of the objective value between two successive estimates.
- *maxit_l1* : Used as stopping criterion for the proximal L1 operator. Maximum number of iterations.
- *param.nu_l1* : bound on the norm^2 of the operator Psi, i.e.

$$\|\Psi x\|^2 \leqslant \nu \|x\|^2$$

- *param.tight_l1* : 1 if Psit is a tight frame or 0 if not (default = 1)
- *param.weights* : weights (default = 1) for a weighted L1-norm defined as:

$$\sum_i w_i |x_i|$$

The problem is solved thanks to a Douglas-Rachford splitting algorithm.
References: [3]

1.3.3 SOLVE_TVDN - Solve TVDN problem

Usage

```
sol = solve_tvdn(y, epsilon, A, At, param)
sol = solve_tvdn(y, epsilon, A, At)
[sol, info] = solve_tvdn(...)
```

Input parameters

y	Measurements
epsilon	Radius of the L2 ball
A	Operator
At	Adjoint of A
param	Optional parameter

Output parameters

sol	Solution
info	Structure summarizing informations at convergence

Description

`sol = solve_tvdn(Y, epsilon, A, At, PARAM)` solves:

$$arg\min_x \|x\|_{TV} s.t. \|y - Ax\|_2 < \varepsilon$$

Y contains the measurements. A is the forward measurement operator and At the associated adjoint operator. PARAM a Matlab structure containing the following fields:

General parameters:

- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.useGPU* : Use GPU to compute the TV prox operator. Please prior call init_gpu and free_gpu to launch and release the GPU library (default: 0).
- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t) - n(t-1)}{n(t)} < tol,$$

  where $n(t) = ||(x)||_{TV}$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.gamma* : control the converge speed (default: 1e-1).

Projection onto the L2-ball :

- *param.tight_b2* : 1 if A is a tight frame or 0 if not (default = 1)
- *param.nu_b2* : bound on the norm of the operator A, i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

- *param.tol_b2* : tolerance for the projection onto the L2 ball (default: 1e-3):

$$\frac{\varepsilon}{1-tol} \leqslant \|y-Az\|_2 \leqslant \frac{\varepsilon}{1+tol}$$

- *param.maxit_b2* : max. nb. of iterations for the projection onto the L2 ball (default 200).

Proximal TV operator:

- *param.maxit_tv* : Used as stopping criterion for the proximal TV operator. Maximum number of iterations.

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the objectivs functions
- *info.crit* : Stopping critterion used
- *info.rel_norm* : Relative norm at convergence
- *info.residue* : Final residue

The problem is solved thanks to a Douglas-Rachford splitting algorithm.
References: [3]

1.4 Demo solver

1.4.1 DEMO_FORWARD_BACKWARD_ALG - Demonstration to define a personal solver

Usage

```
: param.algo = demo_forward_backward_alg();
```

Description

This function returns a structure containing the algorithm. You can lauch your personal algorithm with the following:

```
param.algo = demo_forward_backward_alg();
sol = solvep(x0, {f1, f2}, param);
```

Chapter 2

Unlocbox - Proximal operators

2.1 General Proximal operators

2.1.1 PROX_L0 - Proximal operator of the L0 norm

Usage

```
sol = prox_l0(x)
sol = prox_l0(x, gamma)
sol = prox_l0(x, gamma, param)
[sol, info] = prox_l0(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing information at convergence

Description

`prox_l0(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_0$$

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.k* : number of non zero elements (if not defined, it uses gamma to dertermine how many coefficients are kept

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of execution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping criterion used

2.1.2 PROX_L1 - Proximal operator with L1 norm

Usage

```
sol=prox_l1(x, gamma)
sol=prox_l1(x, gamma, param)
[sol, info]=prox_l1(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_l1(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|Az-y\|_1$$

param is a Matlab structure containing the following fields:

- *param.A* : Forward operator (default: Id).
- *param.At* : Adjoint operator (default: Id).
- *param.y* : y
- *param.tight* : 1 if A is a tight frame or 0 if not (default = 0)
- *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu\|x\|^2$$

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.

- *param.maxit* : max. nb. of iterations (default: 200).
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.weights* : weights for a weighted L1-norm (default = 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

We implemented the algo of "M.J. Fadili and J-L. Starck, "Monotone operator splitting for optimization problems in sparse recovery" see references. See lemma 2 (section 3). The parameter nu is changed to nu^{-1}.

References: [7], [1], [8]

2.1.3 PROX_L2 - Proximal operator with L2 norm

Usage

```
sol=prox_l2(x, gamma)
sol=prox_l2(x, gamma, param)
[sol, info]=prox_l2(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_l2(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma \|w(Az-y)\|_2^2$$

where w are some weights.

param is a Matlab structure containing the following fields:

- *param.weights* : weights for a weighted L2-norm (default = 1)
- *param.y* : measurements (default: 0).
- *param.A* : Forward operator (default: Id).

- *param.At* : Adjoint operator (default: A).
- *param.tightT* : 1 if A^T is a tight frame or 0 if not (default = 0) Note that A^T tight means $AA^T = \nu I$.
- *param.tight* : 1 if A is a tight frame or 0 if not (default = 0) Note that A tight means $A^T A = \nu I$.
- *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t) - n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x - z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.pcg* : Use the fast PCG algorithm (default 1).
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

2.1.4 PROX_L2grad - Proximal operator of the 2 norm of the gradient in 1 dimension

Usage

```
sol=prox_l2grad(x, gamma)
sol=prox_l2grad(x, gamma, param)
[sol, info]=prox_l2grad(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
infos	Structure summarizing informations at convergence

Description

This function compute the 1 dimensional proximal operator of x. For matrices, the function is applied to each column. To use the 2D proximal operator just set up the parameter *param.2d* to 1.
`prox_l2grad(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|\nabla A z\|_2^2$$

param is a Matlab structure containing the following fields:

- ***param.abasis*** (**to use another basis than the DFT (default: 0). To be**) done -- Not working yet
- *param.weights* : weights if you use a an array.
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.d2* : 2 dimentional gradient (default 0)
- *param.A* : Forward operator (default: Id).
- *param.At* : Adjoint operator (default: Id).
- *param.tight* : 1 if A is a tight frame or 0 if not (default = 1)
- *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu\|x\|^2$$

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.deriveorder* : Order ot the derivative default 1

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

2.1.5 PROX_L2gradfourier - Proximal operator of the 2 norm of the gradient in the Fourier domain

Usage

```
sol=prox_l2gradfourier(x, gamma)
sol=prox_l2gradfourier(x, gamma, param)
[sol, info]=prox_l2gradfourier(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

This function compute the 1 dimensional proximal operator of x. For matrices, the function is applied to each column. The parameter param.d2 allows the user to use the 2 dimentional gradient.

Warning: the signal should not be centered. Indice 1 for abscissa 0.

`prox_l2gradfourier(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|\nabla\mathscr{F}z\|_2^2$$

param is a Matlab structure containing the following fields:

- *param.weights* : weights if you use a an array.
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.deriveorder* : Order ot the derivative default 1
- *param.d2* : 2 dimentional gradient (default 0)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

2.1.6 PROX_LINF1 - Proximal operator with L1inf norm

Usage

```
sol = prox_linf1(x, gamma, param)
[sol,info] = prox_linf1(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of parameters (optional)

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_Linf1(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|Ax\|_{\infty 1}$$

The easiest way to use this proximla operator is to give a matrix x as imput. In this case, the sup norm will be computed over the lines (2nd dimention) and the one norm will be computed over the rows (1st dimention).

param is a Matlab structure containing the following fields:

- *param.weights1* : weights for a weighted L1inf-norm works on the norm L1 (default = 1) (Experimental)
- *param.weights2* : weights for a weighted L1inf-norm works on the sup nom (default = 1) (Experimental)
- *param.g_d*, *param.g_t* are the group vectors. If you give a matrix, do not set those parameters.

  param.g_d contains the indices of the elements to be grouped and *param.g_t* the size of the different groups.

  Warning: *param.g_d* and *param.g_t* have to be row vector!

  Example: suppose x=[x1 x2 x3 x4 x5 x6]

  and Group 1: [x1 x2 x4 x5] group 2: [x3 x6]

  In matlab:

  ```
  param.g_d = [1 2 4 5 3 6]; param.g_t=[4 2];
  ```

  Also this is also possible:

  ```
  param.g_d = [4 5 3 6 1 2]; param.g_t=[2 4];
  ```

- *param.multi_group*: in order to group component in a not disjoint manner, it is possible to use the multi_group option. *param.multi_group* is now set automatically by the function.

  Overlaping group: In order to make overlapping group just give a vector of g_d, g_b and g_t. Example:

  ```
  param.g_d=[g_d1; g_d2; ...; g_dn];
  param.g_t=[g_t1; g_t2; ...; g_tn];
  ```

  Warning! There must be no overlap in *g_d1*, *g_d2*,... *g_dn*

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *infos.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [9]

2.1.7 PROX_L21 - Proximal operator with L21 norm

Usage

```
sol=prox_l21(x, gamma, param)
[sol,info] = prox_l21(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_L21(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|x\|_{2,1}$$

where

$$\|x\|_{2,1} = \sum_j \left| \sum_i |x(i,j)|^2 \right|^{1/2}$$

The easiest way to use this proximal operator is to give a matrix x as input. In this case, the $l_{2,1}$ norm is computed like in the expression above.

param is a Matlab structure containing the following fields:

- *param.weights1* : weights for a weighted L21-norm works on the norm L1 (default = 1) (Experimental)
- *param.weights2* : weights for a weighted L21-norm works on the L2 norm (default = 1) (Experimental)
- *param.g_d*, *param.g_t* are the group vectors. If you give a matrix, do not set those parameters.

  param.g_d contains the indices of the elements to be grouped and *param.g_t* the size of the different groups.

  Warning: *param.g_d* and *param.g_t* have to be row vector!

  Example: suppose x=[x1 x2 x3 x4 x5 x6]

  and Group 1: [x1 x2 x4 x5] group 2: [x3 x6]

  In matlab:

  ```
  param.g_d = [1 2 4 5 3 6]; param.g_t=[4 2];
  ```

  Also this is also possible:

  ```
  param.g_d = [4 5 3 6 1 2]; param.g_t=[2 4];
  ```

- *param.multi_group*: in order to group component in a not disjoint manner, it is possible to use the multi_group option. *param.multi_group* is now set automatically by the function.

  Overlaping group: In order to make overlapping group just give a vector of g_d, g_b and g_t. Example:

  ```
  param.g_d=[g_d1; g_d2; ...; g_dn];
  param.g_t=[g_t1; g_t2; ...; g_tn];
  ```

  Warning! There must be no overlap in *g_d1*, *g_d2*,... *g_dn*

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [9], [10], [11], [12]

2.1.8 PROX_L12 - Proximal operator with L12 norm

Usage

```
sol=prox_l12(x, gamma, param)
[sol,info] = prox_l12(x, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_L12(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_{1,2}^2$$

where

$$\|x\|_{1,2}^2 = \sqrt{\sum_j \left|\sum_i |x(i,j)|\right|^2}$$

The easiest way to use this proximal operator is to give a matrix x as input. In this case, the $l_{1,2}$ norm is computed like in the expression above.

param is a Matlab structure containing the following fields:

- *param.weights* : weights for a weighted L12 norm (default = 1)
- *param.g_d*, *param.g_t* are the group vectors. If you give a matrix, do not set those parameters.

  param.g_d contains the indices of the elements to be grouped and *param.g_t* the size of the different groups.

  Warning: *param.g_d* and *param.g_t* have to be row vector!

  Example: suppose x=[x1 x2 x3 x4 x5 x6]

  and Group 1: [x1 x2 x4 x5] group 2: [x3 x6]

  In matlab:

  ```
  param.g_d = [1 2 4 5 3 6]; param.g_t=[4 2];
  ```

  Also this is also possible:

  ```
  param.g_d = [4 5 3 6 1 2]; param.g_t=[2 4];
  ```

- *param.multi_group*: in order to group component in a not disjoint manner, it is possible to use the multi_group option. *param.multi_group* is now set automatically by the function.

  Overlaping group: In order to make overlapping group just give a vector of g_d, g_b and g_t. Example:

  ```
  param.g_d=[g_d1; g_d2; ...; g_dn];
  param.g_t=[g_t1; g_t2; ...; g_tn];
  ```

  Warning! There must be no overlap in *g_d1*, *g_d2*,... *g_dn*

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [9], [10], [11], [12]

2.1.9 PROX_NUCLEARNORM - Proximal operator with the nuclear norm

Usage

```
sol=prox_nuclearnorm(x, gamma)
sol=prox_nuclearnorm(x, gamma, param)
[sol,info]=prox_nuclearnorm(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_NuclearNorm(x, gamma, param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 + \gamma||z||_*$$

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.svds* : 0 uses svd, 1 uses svds. (default: 1 for sparse matrices, 0 for full matrices)
- *param.max_rank* : upper bound of rank expected after thresholding. If actual rank is greater, SVDS has to restart with bigger bound. (default: the maximum between 20 and sqrt(n))
- *param.tol* : tolerance for svds. Bigger tolerance yelds faster results. (default: 1e-5);
- *param.single* : single precision (1) or not (0)? (default: single only if input is single precision);

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping criterion used
- *info.rank* : Rank of the final solution (-1 means the rank was not computed)

2.1.10 PROX_NUCLEARNORM_BLOCK - Proximal operator of nuclear norms of blocks

Usage

```
sol = prox_nuclearnorm_block(x, gamma, ind_r, ind_c)
sol = prox_nuclearnorm_block(x, gamma, ind_r, ind_c, param)
[sol, info] = prox_nuclearnorm_block(...)
```

Input parameters

X	Input matrix
gamma	Regularization parameter
ind_r	Vector partitioning the rows of X in groups EXAMPLE: ind_r [1 2 2 3 3 1] means that the first block contains the first and last rows of x
ind_c	Vector partitioning the columns in groups (same as ind_r)
param	Structure of optional parameters

Output parameters

sol	Solution
info	Structure summarizing information at convergence

Description

`prox_NuclearNorm_Block(x, gamma, param)` solves:

$$sol = arg\min_{Z} \frac{1}{2}\|X-Z\|_F^2 + \sum_{i,j} \gamma W_{i,j} ||Z_{i,j}||_*$$

where Z(i,j) is the i,j-th block indicated by the indices ind_r == i, ind_c == j and w(i,j) is an optional weight for the block

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print info for each block (default: 1)
- *param.single* : single precision (1) or not (0)? (default: single only if input is single precision);
- *param.compute_stat* : if true, the statistics nz_blocks, rank_block, norm_block will be returned as fields of the struct info.
- *param.W* : weight for the term of each block in form of an array.

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of execution of the function in sec.
- *info.final_eval* : Final evaluation of the sum of nuclear norms
- *info.crit* : Stopping criterion used
- *info.rank* : Rank of the final solution (-1 means the rank was not computed)
- *info.nz_blocks* : total number of zero blocks
- *info.rank_block* : array containing the rank of each block
- *info.norm_block* : array containing the nuclear norm of each block

2.1.11 PROX_TV - Total variation proximal operator

Usage

```
sol=prox_tv(x, gamma)
sol=prox_tv(x, gamma,param)
[sol, info]=prox_tv(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

This function compute the 2 dimentional TV proximal operator evaluated in b. If b is a cube, this function will evaluate the TV proximal operator on each image of the cube. For 3 dimention TV proximal operator the function prox_tv3d can be used.

`prox_tv(y, gamma, param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_{TV}$$

param is a Matlab structure containing the following fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.

- *param.maxit* : max. nb. of iterations (default: 200).

- *param.useGPU* : Use GPU to compute the TV prox operator. Please prior call init_gpu and free_gpu to launch and release the GPU library (default: 0).

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

- *param.weights* : weights for each dimention (default $[1,1]$)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used

- *info.iter* : Number of iteration

- *info.time* : Time of exectution of the function in sec.

- *info.final_eval* : Final evaluation of the function

- *info.crit* : Stopping critterion used

References: [13]

2.1.12 PROX_TV3D - Total variation proximal operator

Usage

```
sol=prox_tv3d(x, gamma)
sol=prox_tv3d(x, gamma,param)
[sol, info]=prox_tv3d(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

This function compute the 3 dimentional TV proximal operator evaluated in b. If b is 4 dimentional, this function will evaluate the TV proximal operator on each cube. For 2 dimention TV proximal of cubes operator the function prox_tv can be used.

`prox_tv3d(y, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|x\|_{TV}$$

param is a Matlab structure containing the following fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

  $$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.parrallel* : Parallelisation level. 0 means no parallelization, 1 means all cubes (fourth dimension changing) at the same time.
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.useGPU* : Use GPU to compute the TV prox operator. Please prior call init_gpu and free_gpu to launch and release the GPU library (default: 0).
- *param.weights* : weights for each dimention (default $[1,1,1]$)

infos is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [13]

2.1.13 PROX_TV1D - Total variation proximal operator

Usage

```
sol=prox_tv1d(x, gamma)
sol=prox_tv1d(x, gamma, param)
[sol, info]=prox_tv1d(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing information at convergence

Description

This function computes the 1 dimensional TV proximal operator evaluated in b. If b is a matrix, this function will evaluate the TV proximal operator on each row of the matrix. For 2D, TV proximal operator `prox_tv` can be used.

`prox_tv(y, gamma, param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|x\|_{TV}$$

param is a Matlab structure containing the following fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

  $$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.use_fast* : Use the fast algorithm of Laurent Condat.
- *param.useGPU* : Use GPU to compute the TV prox operator. Please prior call init_gpu and free_gpu to launch and release the GPU library (default: 0).
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [14], [13]

2.1.14 PROX_TV4D - Total variation proximal operator

Usage

```
sol=prox_tv4d(x, gamma)
sol=prox_tv4d(x, gamma,param)
[sol, info]=prox_tv4d(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

This function compute the 4 dimentional TV proximal operator evaluated in b. If b is 5 dimentional, this function will evaluate the TV proximal operator on each 4 dimentional cube.
`prox_tv4d(y, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|x\|_{TV}$$

param is a Matlab structure containing the following fields:

- *param.tol* : is stop criterion for the loop. The algorithm stops if

  $$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.parrallel* : Parallelisation level. 0 means no parallelization, 1 means working on all the data at once
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.weights* : weights for each dimention (default $[1,1,1,1]$)
- *param.useGPU* : Use GPU to compute the TV prox operator. Please prior call init_gpu and free_gpu to launch and release the GPU library (default: 0).

infos is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.

- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

References: [13]

2.1.15 PROX_SUM_LOG - Proximal operator of log-barrier - sum(log(x))

Usage

```
sol = prox_sum_log(x, gamma)
sol = prox_sum_log(x, gamma, param)
[sol, info] = prox_sum_log(x, gamma, param)
```

Input parameters

x	Input signal (vector or matrix!).
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_sum_log(x, gamma, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 - \gamma \sum_i (log(z_i))$$

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, (1) print -sum(log(z)), 2 additionally report negative inputs.

MATRICES: Note that this prox works for matrices as well. The log of the sum gives the same result independently of which dinension we perform the summation over:

```
sol = (x + sqrt(x.^2 + 4*gamma)) /2;
```

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

2.1.16 PROX_SUM_LOG_NORM2 - Proximal operator of log-barrier - sum(log(x))

Usage

```
sol = prox_sum_log_norm2(x, alpha, beta, gamma)
sol = prox_sum_log_norm2(x, alpha, beta, gamma, param)
[sol, info] = prox_sum_log_norm2(x, alpha, beta, gamma, param)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
alpha	multiplier of -log
beta	multiplier of norm-2
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_l1(x, gamma, param)` solves:

$$sol = \arg\min_z \frac{1}{2}\|x-z\|_2^2 - \gamma(\alpha\sum_i(log(z_i)) + \beta/2\|z\|_2^2)$$

param is a Matlab structure containing the following fields:

- ***param.verbose*** (**0 no log, (1) print -sum(log(z)), 2 additionally**) report negative inputs.

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

2.2 Projection operators

2.2.1 PROJ_B1 - Projection onto a L1-ball

Usage

```
sol=proj_b1(x, ~, param)
[sol,infos]=proj_b1(x, ~, param)
```

Input parameters

x	Input signal.
param	Structure of parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`proj_b1(x,~,param)` solves:

$$sol = \min_z ||x-z||_2^2 \qquad s.t. \qquad \|w.*z\|_1 < \varepsilon$$

Remark: the projection is the proximal operator of the indicative function of $||w.*z||_1 < \varepsilon$. So it can be written:

$$prox_{f,\gamma}(x) \qquad where \qquad f = i_c(\|w.*z\|_1 < \varepsilon)$$

param is a Matlab structure containing the following fields:

- *param.epsilon* : Radius of the L1 ball (default = 1).
- *param.weight* : contain the weights (default ones).
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

Rem: The input "~" is useless but needed for compatibility issue.
This code is partly borrowed from the SPGL toolbox!

2.2.2 PROJ_B2 - Projection onto a L2-ball

Usage

```
sol=proj_b2(x, ~, param)
[sol, infos]=proj_b2(x, ~, param)
```

Input parameters

x	Input signal.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`proj_b2(x,~,param)` solves:

$$sol = arg\min_z ||x-z||_2^2 \qquad s.t. \qquad ||y-Az||_2 <= \varepsilon$$

Remark: the projection is the proximal operator of the indicative function of $||y-Az||_2 < \varepsilon$. So it can be written:

$$prox_{f,\gamma}(x) \qquad where \qquad f = i_c(\|y-Az\|_2 <= \varepsilon)$$

param is a Matlab structure containing the following fields:

- *param.y* : measurements (default: 0).
- *param.A* : Forward operator (default: Id).
- *param.At* : Adjoint operator (default: Id).
- *param.epsilon* : Radius of the L2 ball (default = 1e-3).
- *param.tight* : 1 if A is a tight frame or 0 if not (default = 0)
- *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

- *param.tol* : tolerance for the projection onto the L2 ball (default: 1e-3) . The algorithms stops if

$$\frac{\varepsilon}{1-tol} \leqslant \|y-Az\|_2 \leqslant \frac{\varepsilon}{1+tol}$$

- *param.maxit* : max. nb. of iterations (default: 200).
- ***param.method*** (**is the method used to solve the problem. It can be 'FISTA' or**) 'ISTA'. By default, it's 'FISTA'.
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of execution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used
- *info.residue* : Final residue

Rem: The input "~" is useless but needed for compatibility issue.
References: [7]

2.2.3 PROJ_BOX - Projection onto the box set (multidimensional interval constraint)

Usage

```
sol = proj_box(x, [])
sol = proj_box(x)
sol = proj_box(x, [], param)
[sol, info] = proj_box(x, [], param)
```

Input parameters

x	Input signal.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing information at convergence

Description

`prox_box(x, [], param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 \text{ subject to } z < z_{max} \text{ and } z > z_{min}$$

where `zmax` and `zmin` might be scalar or vector valued.
param is a Matlab structure containing the following fields:

- *param.lower_lim* : lower bound(s) for z (default 0)
- *param.upper_lim* : upper bound(s) for z (default 1)

if these two are vector-valued, bounds apply entry-by-entry

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iterations (this function is not iterative)
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used (one shot here)

Rem: The input "~" is useless but needed for compatibility issue.

2.2.4 PROJ_NUCLEARNORM - Projection on the nuclear norm ball

Usage

```
sol=proj_nuclearnorm(x);
sol=proj_nuclearnorm(x, gamma, param);
[sol,info]=proj_nuclearnorm(...);
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`proj_nuclearnorm(x, gamma, param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 \text{ s. t. } ||z||_* < \varepsilon$$

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.epsilon* : Radius of the nuclear ball (default = 1).
- *param.svds* : 0 uses svd, 1 uses svds. (default: 1 for sparse matrices, 0 for full matrices)
- *param.max_rank* : upper bound of rank expected after thresholding. If actual rank is greater, SVDS has to restart with bigger bound. (default: the maximum between 20 and sqrt(n))
- *param.tol* : tolerance for svds. Bigger tolerance yelds faster results. (default: 1e-5);
- *param.single* : single precision (1) or not (0)? (default: single only if input is single precision);

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping criterion used
- *info.rank* : Rank of the final solution (-1 means the rank was not computed)

2.2.5 PROJ_SPSD - Projection on the Symetric positive semi definite set of matrices

Usage

```
sol=proj_spsd(x)
sol=proj_spsd(x, 0, param)
[sol,info]=proj_spsd(...)
```

Input parameters

x	Input signal.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`proj_spsd(x, gamma, param)` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 \text{ s. t. } x \text{ is SDSD}$$

param is a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping criterion used

2.2.6 PROJ_LINEAR_EQ - projection onto the space Az = y

Usage

```
sol = proj_linear_eq(x, ~, param)
[sol, info] = proj_linear_eq(x, ~, param)
```

Input parameters

x	Input signal.
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`proj_linear_eq(x,~,param)` solves:

$$sol = \min_z ||x-z||_2^2 \qquad s.t. \qquad Az = y$$

param is a Matlab structure containing the following fields:

- *param.y* : vector (default: 0).
- *param.method* : method used 'exact' or 'iterative' (default: 'exact').
- *param.A* : Matrix A (default: Id) (Or operator for the 'iterative' method)
- *param.At* : Matrix or operator At (Only for the 'iterative' method)
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.nu* : (only for iterative method) bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

- *param.pinvA* : $A*(AA^*)^(-1)$ Pseudo inverse of A Define this parameter to speed up computation (Only for 'exact').

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of execution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

Rem: The input "~" is useless but needed for compatibility issue.

2.2.7 PROJ_LINEAR_INEQ - projection onto the space Az = y

Usage

```
sol = proj_linear_ineq(x, ~, param)
[sol, infos] = proj_linear_ineq(x, ~, param)
```

Input parameters

x	Input signal.
param	Structure of optional parameters.

Output parameters

sol	Solution.
infos	Structure summarizing informations at convergence

Description

`proj_linear_ineq(x,~,param)` solves:

$$sol = \min_z ||x-z||_2^2 \qquad s.t. \qquad Az \leqslant y$$

param is a Matlab structure containing the following fields:

- *param.y* : vector (default: 0).
- *param.method* : method used 'quadprog' or 'iterative' (default: 'quadprog').
- *param.A* : Matrix A (default: Id) (Or operator for the 'iterative' method)
- *param.At* : Matrix or operator At (Only for the 'iterative' method)
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.nu* : (only for iterative method) bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu \|x\|^2$$

infos is a Matlab structure containing the following fields:

- *infos.algo* : Algorithm used
- *infos.iter* : Number of iteration
- *infos.time* : Time of execution of the function in sec.
- *infos.final_eval* : Final evaluation of the function
- *infos.crit* : Stopping critterion used

Rem: The input "~" is useless but needed for compatibility issue.

2.3 Proximal tools

2.3.1 PROX_sumG - Proximal operator of a sum of function

Usage

```
sol=prox_sumg(x, gamma, param)
[sol, info]=prox_sumg(...)
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
param	Structure of parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`prox_sumG(x, gamma , param)` solves:

$$sol = \arg\min_z \frac{1}{2}\|x-z\|_2^2 + \gamma \sum_i w_i G_i(z) \qquad for \qquad z,x \in R^N$$

param is a Matlab structure containing the following fields:

- *param.G* : cellarray of structure with all the prox operator inside and eventually the norm if no norm is defined, the L^1 norm is used the prox: *F{i}.prox* and norm: *F{i}.eval* are defined in the same way as in the Forward-backward and Douglas-Rachford algorithms
- *param.weights* : weights of different functions (default = $1/N$, where N is the total number of function)
- *param.tol* : is stop criterion for the loop. The algorithm stops if

  $$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f_1(Lx) + f_2(x)$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.lambda_t*: is the weight of the update term. By default 1. This should be between 0 and 1.
- *param.maxit* : is the maximum number of iteration. By default, it is 200.
- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps.

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

Demo: demo_prox_multi_functions
References: [5]

2.3.2 PROX_ADJOINT - Proximal operator of the adjoint function of f

Usage

```
sol=prox_adjoint(x, gamma, f);
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
f	Function

Output parameters

sol	Solution.
infos	Structure summarizing informations at convergence

Description

‘ prox_adjoint(x,gamma,f)‘ solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma f^*$$

where f^* is the adjoint of f. This problem is solved thanks to the Moreau's identity.

Warning: f needs to be a proper convex lower semi continuous function.

2.3.3 PROX_ADD_2NORM - Proximal operator with an additional quadratic term

Usage

```
sol = prox_add_2norm(x, gamma, param);
```

Input parameters

x	Input signal.
gamma	Regularization parameter.
f	Function

Output parameters

sol	Solution.
infos	Structure summarizing informations at convergence

Description

`prox_add_2norm( x,gamma,param )` solves:

$$sol = arg\min_z \frac{1}{2}\|x-z\|_2^2 + \frac{1}{2}\|y-z\|_2^2 + \gamma f(z)$$

This problem can be solved because we have the nice relationship

$$\frac{1}{2}\|x-z\|_2^2 + \frac{1}{2}\|y-z\|_2^2 = \|\frac{x+y}{2} - z\|_2^2 + \frac{1}{4}\|y-x\|_2^2$$

This can be used to reduce the number of functionals and the solution is

$$sol = prox_{\gamma/2f}\left(\frac{x+y}{2}\right)$$

param is a Matlab structure containing the following fields:

- *param.y* : a vector of the same size as x
- *param.f* : a structure containing the function f

2.3.4 PROX_FAX - Proximal operator of the adjoint function of f

Usage

```
sol=prox_adjoint(x, gamma, param);
```

Input parameters

x	Input signal.
gamma	Regularization parameter (ususally it should be 1)
param	Parameter (Please see: param.f)

Output parameters

sol	Solution.
infos	Structure summarizing informations at convergence

Description

‘ prox_fax(x,gamma,param)‘ solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma f(Ax)$$

This method allows to compute the proximal operator of f(Ax) when only the proximal operator of A can be computed. This function use an ADMM splitting.

param is a non optional structure of parameter containing 2 mendatory parameter:

- *param.A* : Forward operator
- *param.At* : Adjoint operator
- *param.f* : is a structure representing a convex function. Inside the structure, there have to be the prox of the function that can be called by *f1.prox* and the function itself that can be called by *f1.eval*.

As an option, you may specify

- *param.tight* : 1 if A is a tight frame or 0 if not (default = 1)
- *param.nu* : bound on the norm of the operator A (default: 1), i.e.

$$\|Ax\|^2 \leqslant \nu\|x\|^2$$

- *param.tol* : is stop criterion for the loop. The algorithm stops if

$$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-4`.
- *param.maxit* : max. nb. of iterations (default: 200).
- *param.L2_maxit* : max. nb. of iterations for the l2 proximal operator (default: 30).
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

Chapter 3

UnLocBoX - Demos

3.1 Tutorial demos

3.1.1 DEMO_UNLOCBOX - Simple tutorial for the UNLocBoX

Description

Welcome to the tutorial of the UNLocBoX. In this document, we provide an example application that uses the basic concepts of the toolbox. Here you will also find some tricks that may be very useful. You can find an introduction and more detailed documentation in the userguide, available at http://unlocbox.sourceforge.net/notes/unlocbox-note-002.pdf

This toolbox is designed to solve convex optimization problems of the form:

$$arg \min_{x \in \mathbb{R}^N} \left(f_1(x) + f_2(x) \right),$$

or more generally

$$arg \min_{x \in \mathbb{R}^N} \sum_{n=1}^{K} f_n(x),$$

where the f_i are lower semi-continuous convex functions and x the optimization variables. For more details about the problems, please refer to the userguide (UNLocBoX-note-002) available on https://lts2.epfl.ch/unlocbox/notes/unlocbox-note-002.pdf .

This toolbox is based on proximal splitting methods. Those methods cut the problem into smaller (and easier) subproblems that can be solved in an iterative fashion. The UNLocBoX essentially consists of three families of functions:

- Proximity operators: they solve small minimization problems and allow a quick implementation of many composite problems.
- Solvers: generic minimization algorithms that can work with different combinations of proximity operators in order to minimize complex objective functions
- Demonstration files: examples to help you to use the toolbox

This toolbox is provided for free. We would be happy to receive comments, information about bugs or any other kind of help in order to improve the toolbox.

Figure 3.1: The original image provided by the toolbox. Use cameraman() function to access.

A simple example: Image in-painting

Let's suppose we have a noisy image with missing pixels. Our goal is simply to fill the unknown values in order to reconstruct an image close to the original one. We first begin by setting up some assumptions about the problem.

Assumptions

In this particular example, we firstly assume that we know the position of the missing pixels. This happens when we know that a specific part of a photo is destroyed, or when we have sampled some of the pixels in known positions and we wish to recover the rest of the image. Secondly, we assume that the image follows some standard distribution. For example, many natural images are known to have sharp edges and almost flat regions (the extreme case would be the cartoon images with completely flat regions). Thirdly, we suppose that known pixels are subject to some Gaussian noise with a variance of ε.

Formulation of the problem

At this point, the problem can be expressed in a mathematical form. We will simulate the masking operation with an operator A. This first assumption leads to a constraint.

$$Ax = y$$

where x is the vectorized image we want to recover, y are the observed noisy pixels and A a linear operator selecting the known pixels. However due to the addition of noise this constraint can be a little bit relaxed and we rewrite it in the following form

$$\|Ax - y\|_2 \leqslant \sqrt{N}\varepsilon$$

where N is the number of known pixels. Note that ε can be chosen to be equal to 0 so that the equality $y = Ax$ is satisfied. In our case, as the measurements are noisy, we set ε to be the expected value of the norm of the noise

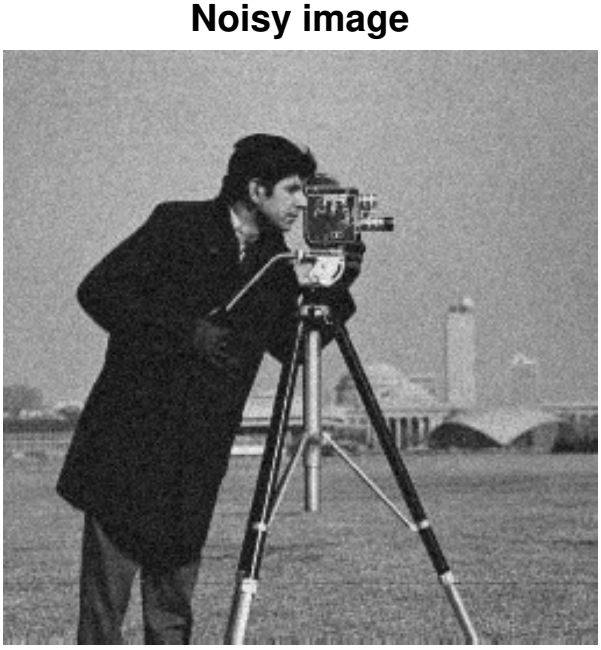


Figure 3.2: Noisy image.

Figure 3.3: Measurements. 50 percent of the pixels have been removed.

(standard deviation times square root of number of measurements).

We use as a prior assumption that the image has a small total variation norm (TV-norm). (The TV-norm is the l^1-norm of the gradient of x.) On images, this norm is low when the image is composed of patches of color and few "degradee" (gradients). This is the case for most of natural images. To summarize, we express the problem as

$$arg\min_x \|x\|_{TV} \qquad \text{subject to} \qquad \|Ax-y\|_2 \leqslant \sqrt{N}\varepsilon \qquad \text{(Problem I)}$$

Note that if the amount of noise is not known, epsilon as a free parameter that tunes the confidence to the measurements. However, this is not the only way to define the problem. We could also write:

$$arg\min_x \|Ax-y\|_2^2 + \lambda \|x\|_{TV} \qquad \text{(Problem II)}$$

with the first function playing the role of a data fidelity term and the second a prior assumption on the signal. λ adjusts the tradeoff between measurement fidelity and prior assumption. We call it the `regularization parameter`. The smaller it is, the more we trust the measurements and conversely. ε plays a similar role as λ.

We have presented two ways to formulate the problem. The reader should keep in mind that choosing between one or the other problem will affect the choice of the solver and the convergence rate. With experience, one should be able to know in advance which problem will lead to the best solver.

Note that there exists a bijection between the parameters λ and ε leading both problems to the same solution. Unfortunately, the bijection function is not trivial to determine.

Once your problem is well defined, we need to provide a list of functions to the UNLocBoX solver. (For example, in Problem 2, the functions are $\|Ax-y\|_2^2$ and :math:‘ lambda |x|_{TV}‘.) Every function is modeled by a MATLAB structure containing some special fields. We separate the functions in two different types: differentiable and non differentiable. For differentiable function, the user needs to fill the following fields: * *func.eval* : An anonymous function that evaluate the function * *func.grad* : An anonymous function that evaluate the gradient * *func.beta* : An upper bound on the Lipschitz constant of the gradient

For instance, the function $\|Ax-y\|_2^2$ is defined in MATLAB by:

```
fsmooth.grad = @(x) 2 * A' * (A*x - y);
fsmooth.eval = @(x) norm(A*x - y)^2;
fsmooth.beta = 2 * norm(A)^2;
```

The Lipschitz constant of a the gradient is defined as:

$$\min_\beta \text{ s.t } \forall x_1, x_2 \in \mathbb{R}^N \text{ we have } \|\nabla_f(x_1) - \nabla_f(x_2)\|_2 \leqslant \beta \|x_1 - x_2\|_2$$

When the function is not differentiable, the field *.beta* is dropped and *.grad* is replaced by the field *.prox* that contains an anonymous function for the proximity operator (They will be explained in more details the following section.

ftv.prox = @(x, T) prox_tv(x, T * lambda, paramtv); ftv.eval = @(x) lambda * tv_norm(x);

Proximity operators

The proximity operator of a lower semi-continuous convex function f is defined by:

$$prox_{\lambda f}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda f(x)$$

Proximity operators minimize a function without going too far from a initial point. They can be thought or assimilated as de-noising operators. Because of the l2-term in the minimization problem, proximity operators perform a regularized minimization of the function f. However, applied iteratively, they lead to the minimization of this function. For x^* the minimizer of the function f, it is obvious that:

$$x^* = prox_f(x^*) = arg\min_x \frac{1}{2}\|x-x^*\|_2^2 + f(x)$$

In a sense, proximity operators perform a regularized minimization of the function f. However, they also provide a framework to handle constraints. Those can be inserted into the problem thanks to indicative functions. These functions assert if x belong to a set C. They only have two output values: 0 if x is in the set and ∞ otherwise:

$$i_C : \mathbb{R}^L \to \{0, +\infty\} : x \mapsto \begin{cases} 0, & \text{if } \ x \in C \\ \\ +\infty & \text{otherwise} \end{cases}$$

The solution of the proximity operator of this function has to be in the set C, otherwise the $i_C(x) = \infty$. Moreover, since it also minimizes $\|x-z\|_2^2$, it will select the closest point to z. As a result the proximity operators of indicator functions are projections.

It is important to keep in mind the equivalence between constraints and indicative functions. This is the trick that allows to use hard constraint with the UNLocBoX as it cannot directly handle them. The constraints will thus be inserted in the form of indicative functions.

Solving problem I

The UNLocBoX is based on proximal splitting techniques for solving convex optimization problems. These techniques divide the problem into smaller problems that are easier to solve. Topically, each function will compose a sub-problem that will be solved by its proximity operator (or gradient step). In the particular case of problem (I), the solver will iteratively, first minimize a little bit the TV norm and second perform the projection on the fidelity term B2-ball. (The B2-ball is the space of point x satisfying $\|Ax-y\| \leqslant \sqrt{N}\varepsilon$). To solve problem (I), we minimize two functions:

- The TV norm: $f_1(x) = \lambda ||x||_{TV}$ The proximity operator of f_1 is given by:

  $$prox_{f1,\lambda}(x) = arg\min_z \frac{1}{2}\|x-z\|_2^2 + \lambda\|z\|_{TV}$$

  In MATLAB, the function is defined by the following code:

  ```
  paramtv.verbose = 1;
  paramtv.maxit = 50;
  f1.prox = @(x, T) prox_tv(x, T * lambda, paramtv);
  f1.eval = @(x) lambda * tv_norm(x);
  ```

  This function is a structure with two fields. First, *f1.prox* is an operator taking as input x and T and evaluating the proximity operator of the function (T has be stay a free weight for the solver. it is going to be replaced by the timestep later). Second, *f1.eval* is also an operator evaluating the function at x.

  The proximal operator of the TV norm is already implemented in the UNLocBoX by the function `prox_tv`. We tune it by setting the maximum number of iterations and a verbosity level. Other parameters are also available (see documentation).

  - *paramtv.verbose* selects the display level (0 no log, 1 summary at convergence and 2 display all steps).
  - *paramtv.maxit* defines the maximum number of iteration for this proximity operator.

  Not that for problem (I), lambda can be dropped or set to 1. This parameter will be used when solving problem (II).

- f_2 is the indicator function of the set S defined by $||Ax-y||_2 < \varepsilon$ The proximity operator of f_2 is:

$$prox_{f2,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

with $i_S(x)$ is zero if x is in the set S and infinite otherwise. Under some technical assumption, this previous problem has an identical solution as:

$$arg\min_z \|x-z\|_2^2 \qquad \text{subject to} \qquad \|Az-y\|_2 \leqslant \varepsilon$$

It is simply a projection on the B2-ball (The B2-ball is the set of all points satisfying $\|Ax-y\|_2 < \varepsilon$). In MATLAB, we write:

```
param_proj.epsilon = epsilon;
param_proj.A = A;
param_proj.At = A;
param_proj.y = y;
f2.prox=@(x,T) proj_b2(x,T,param_proj);
f2.eval=@(x) eps;
```

The *prox* field of *f2* is in that case the operator computing the projection. Since we suppose that the constraint is satisfied, the value of the indicator function is 0. For implementation reasons, it is better to set the value of the operator *f2.eval* to *eps* than to 0. Note that this hypothesis could lead to strange evolution of the objective function. Here the parameter *A* and *At* are mandatory. Please notice here the two following lines:

```
param_proj.A = A;
param_proj.At = A;
```

In fact we consider here the masking operator *A* as a diagonal matrix containing 1's for observed pixels and 0's for hidden pixels. As a consequence: *A* = *At*. In MATLAB, one easy way to implement this operator is to use:

```
A = @(x) matA .* x;
```

with *matA* the mask. In a compressed sensing problem for instance, you would define:

```
param_proj.A = @(x) Phi * x;
param_proj.At = @(x) Phi' * x;
```

where *Phi* is the sensing matrix!

At this point, we are ready to solve the problem. The UNLocBoX contains many different solvers and also a universal one that will select a suitable method for the problem. To use it, just write:

```
sol = solvep(y,{f1,f2});
```

You can also use a specific solver for your problem. In this tutorial, we present two of them `forward_backward` and `douglas_rachford`. Both of them take as input two functions (they have generalization taking more functions), a starting point and some optional parameters.

In our problem, both functions are not smooth on all points of the domain leading to the impossibility to compute the gradient. In that case, solvers (such as `forward_backward`) using gradient descent cannot be used. As a consequence, we will use `douglas_rachford` instead. In MATLAB, we write:

```
param.verbose = 2;
param.maxit = 50;
param.tol = 10e-5;
param.gamma = 0.1;
fig = figure(100);
param.do_sol=@(x) plot_image(x,fig);
sol = douglas_rachford(y,f1,f2,param);
```

Or in an equivalent manner (this second way is recommended):

```
param.method = "douglas_rachford"
sol = solvep(y,{f1,f2},param);
```

- ***param.verbose*** **selects the display level (0 no log, 1 summary at** convergence and 2 display all steps).
- *param.maxit* defines the maximum number of iteration.
- *param.tol* is stopping criterion for the loop. The algorithm stops if

$$\frac{n(t)-n(t-1)}{n(t)} < tol,$$

  where $n(t)$ is the objective function at iteration t
- *param.gamma* defines the step-size. It is a compromise between convergence speed and precision. Note that if *gamma* is too big, the algorithm might not converge. By default, this parameter is computed automatically.
- Finally, the following line allows to display the current reconstruction of the image at each iteration:

```
param.do_sol=@(x) plot_image(x,fig);
```

You can stop the simulation by typing "ctrl + d" in the consol. At the end of the next iteration, the algorithm will stop and return the current solution.

Solving problem II

Solving problem II instead of problem I can be done with a small modification of the previous code. First we define another function as follow:

```
f3.grad = @(x) 2*A(A(x) - y);
f3.eval = @(x) norm(A(x) - y, 'fro')^2;
f3.beta = 2;
```

The structure of *f3* contains a field *f3.grad*. In fact, the l2-norm is a smooth function. As a consequence the gradient is well defined on the entire domain. This allows using the `forward_backward` solver that can be called by:

```
param.method = "forward_backward"
sol21 = solvep(y,{f1,f2},param);
```

In this case, we can also use the `douglas_rachford` solver. To do so, we need to define the field *f3.prox*. In general, this is not recommended because a gradient step is usually less computationally expensive than a proximal operator:

Figure 3.4: This figure shows the reconstructed image by solving problem I using Douglas Rachford algorithm.

```
param_l2.A = A;
param_l2.At = A;
param_l2.y = y;
param_l2.verbose = 1;
f3.prox = @(x,T) prox_l2(x, T, param_l2);
f3.eval = @(x) norm(A(x) - y, 'fro')^2;

param.method = "douglas_rachford"
sol22 = solvep(y, {f1,f3}, param);
```

We remind the user that `forward_backward` will not use the field *f3.prox* and `douglas_rachford` will not use the field *f3.grad*.

These two solvers will converge (up to numerical error) to the same solution. However, convergence speed might be different. As we perform only 100 iterations with both of them, we do not obtain exactly the same result.

Remark: The parameter *lambda* (the regularization parameter) and *epsilon* (The radius of the l2 ball) can be chosen empirically. Some methods allow to compute those parameters. However, this is far beyond the scope of this tutorial.

Conclusion

In this tutorial, the reader can observe that problem (II) is solved much more efficiently than problem (I).However, writing the problem with a constraint (like problem (I)) often allow a much easier tuning of the parameters at the cost of using a slower solver.

Only experience helps to know which formulation of a problem will lead to the best solver. Usually, forward backward (FISTA) and ADMM are considered to be the best solvers.

Speed consideration are relative when using the UNLocBoX. Due to general implementation of the toolbox, we estimate the overall speed between one and two times slower than an optimal algorithm cooked and optimized for a special problem (in MATLAB).

Figure 3.5: This figure shows the reconstructed image by solving problem II using the Forward Backward algorithm.

Figure 3.6: This figure shows the reconstructed image by solving problem II using the Douglas Rachford algorithm.

Thanks for reading this tutorial
References: [2], [15]

3.2 Practical example of the toolbox

3.2.1 DEMO_COMPRESS_SENSING - Compress sensing example using forward backward algorithm

Description

We present a compress sensing example solved with the forward backward solver. The problem can be expressed as this

$$\arg\min_x \|Ax-b\|^2 + \tau\|x\|_1$$

Where b are the measurements and A the measurement matrix.
We set

- $f_1(x) = ||x||_1$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = \arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_1$$

  This function is simply a soft thresholding.

- $f_2(x) = ||Ax-b||_2^2$ We define the gradient as:

$$\nabla_f(x) = 2A^*(x-b)$$

  A is the measurement matrix (random Gaussian distribution)

The number of measurements M is computed with respect of the size of the signal N and the sparsity level K:

$$M = K\max(4, \text{ceil}(\log(N)))$$

With this number of measurements, the algorithm is supposed to perform very often always a perfect reconstruction. This plot is automatically generated; let's hope it will be the case.

Results

References: [3], [2]

3.2.2 DEMO_COMPRESS_SENSING2 - Compress sensing example using Douglas Rachford algorithm

Description

We present a compress sensing example solved with the douglas rachford solver. The problem can be expressed as this

$$arg\min_x \|x\|_1 \qquad such \quad that \qquad \|b-Ax\|_2 \leqslant \varepsilon$$

Where b are the measurements and A the measurement matrix.
We set

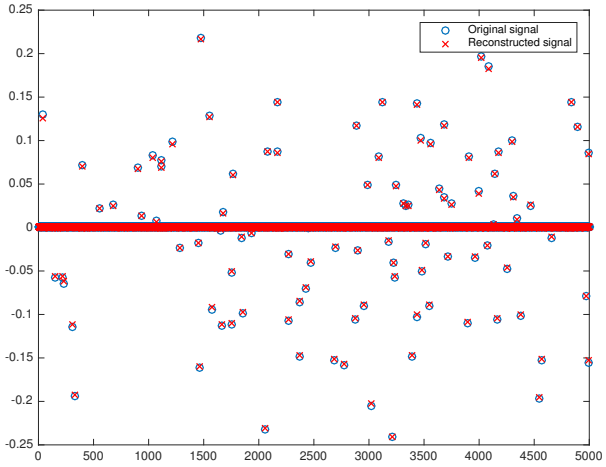


Figure 3.7: Results of the algorithm

This figure shows the original signal and the reconstruction done thanks to the algorithm and the measurements. The number of measurements is M=900, the length of the signal N=5000 and K=100. This is equivalent to a compression ratio of 5.55.

- $f_1(x) = ||x||_1$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_1$$

  This function is simply a soft thresholding.

- f_2 is the indicator function of the set S define by $||Ax-b||_2 < \varepsilon$ We define the prox of f_2 as

$$prox_{f2,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

  with $i_S(x)$ is zero if x is in the set S and infinity otherwise. This previous problem has an identical solution as:

$$arg\min_z \|x-z\|_2^2 \qquad such \quad that \qquad \|Az-b\|_2 \leqslant \varepsilon$$

  It is simply a projection on the B2-ball. A is the measurement matrix (random Gaussian distribution)

The number of measurements M is computed with respect of the size of the signal N and the sparsity level K:

$$M = K\max(4, \text{ceil}(\log(N)))$$

With this number of measurements, the algorithm is supposed to perform very often always a perfect reconstruction. This plot is automatically generated, let's hope it will be the case.

Results

References: [3], [2]

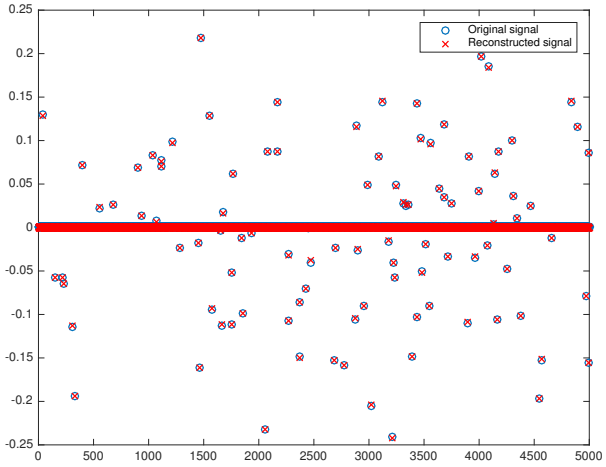


Figure 3.8: Results of the algorithm

This figure shows the original signal and the reconstruction done thanks to the algorithm and the measurements. The number of measurements is M=900, the length of the signal N=5000 and K=100. This is equivalent to a compression ratio of 5.55.

3.2.3 DEMO_COMPRESS_SENSING3 - Compress sensing example using grouped L12 norm

Description

We present a compress sensing example solved with the douglas rachford solver. The particularity of this example is the use of a mixed norm. We do not only know the the signal is sparse, we also know that the sparse coefficients are grouped.

The problem can be expressed as this

$$\arg\min_x \|x\|_{2,1} \qquad such \quad that \qquad \|b-Ax\|_2 \leqslant \varepsilon$$

Where b are the measurements and A the measurement matrix.

We set

- $f_1(x) = ||x||_{2,1}$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = \arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_{2,1}$$

- f_2 is the indicator function of the set S define by $||Ax-b||_2 < \varepsilon$ We define the prox of f_2 as

$$prox_{f2,\gamma}(z) = \arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

with $i_S(x)$ is zero if x is in the set S and infinity otherwise. This previous problem has an identical solution as:

$$\arg\min_z \|x-z\|_2^2 \qquad such \quad that \qquad \|Az-b\|_2 \leqslant \varepsilon$$

It is simply a projection on the B2-ball. A is the measurement matrix (random Gaussian distribution)

The theoretical number of measurements M is computed with respect of the size of the signal N and the sparsity level K:

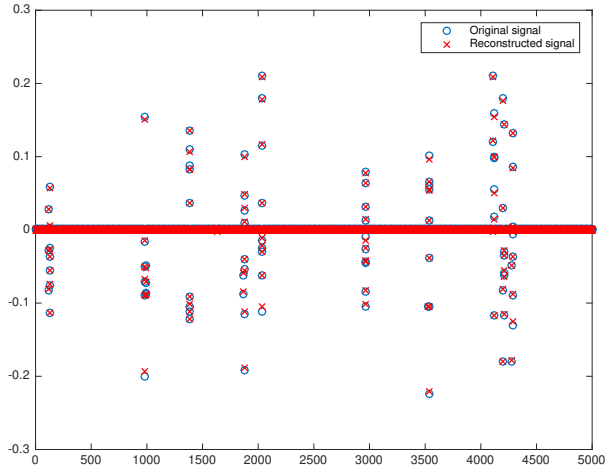


Figure 3.9: Results of the algorithm

This figure shows the original signal and the reconstruction done thanks to the algorithm and the measurements. The number of measurements is M=900, the length of the signal N=5000, K=100, p=4. This is equivalent to a compression ratio of 16.67. The elements are grouped by 10.

$$M = K \max\left(4, \text{ceil}(\log(N))\right).$$

Since we add some new information, we will try to reduce the number of measurements by a factor p:

$$M = K \max\left(\frac{4}{p}, \text{ceil}(\frac{\log(N)}{p})\right).$$

With this number of measurements, we hope that the algorithm will perform a perfect reconstruction.

Results

References: [3], [9], [2]

3.2.4 DEMO_COMPRESS_SENSING4 - Compress sensing example using grouped L1inf norm

Description

We present a compress sensing example solved with the douglas rachford solver. The particularity of this example is the use of a mixed norm. We do not only know that the signal is sparse, but we also know that the sparse coefficients are grouped.

The problem can be expressed as this

$$arg\min_x \|x\|_{1\infty} \qquad such \quad that \qquad \|b - Ax\|_2 \leqslant \varepsilon$$

Where b are the measurements and A the measurement matrix.

We set

- $f_1(x) = ||x||_{1\infty}$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x - z\|_2^2 + \gamma\|z\|_{1\infty}$$

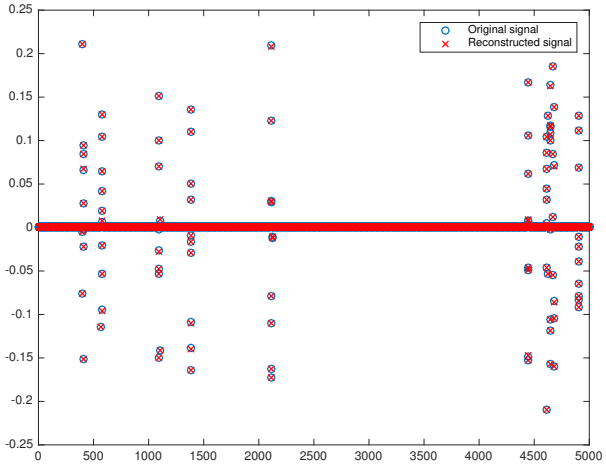


Figure 3.10: Results of the algorithm

This figure shows the original signal and the reconstruction done thanks to the algorithm and the measurements. The number of measurements is M=900, the length of the signal N=5000, K=100, p=2. This is equivalent to a compression ratio of 10. The elements are grouped by 10.

- f_2 is the indicator function of the set S define by $||Ax-b||_2 < \varepsilon$ We define the prox of f_2 as

$$prox_{f2,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

  with $i_S(x)$ is zero if x is in the set S and infinity otherwise. This previous problem has an identical solution as:

$$arg\min_z \|x-z\|_2^2 \qquad such \quad that \qquad \|Az-b\|_2 \leqslant \varepsilon$$

  It is simply a projection on the B2-ball. A is the measurement matrix (random Gaussian distribution)

The theoretical number of measurements M is computed with respect of the size of the signal N and the sparsity level K:

$$M = K\max\left(4, \text{ceil}(\log(N))\right).$$

Since we add some new information, we will try to reduce the number of measurements by a factor p:

$$M = K\max\left(\frac{4}{p}, \text{ceil}(\frac{\log(N)}{p})\right).$$

With this number of measurements, we hope that the algorithm will perform a perfect reconstruction.

Results

References: [3], [9], [2]

3.2.5 DEMO_DECONVOLUTION - Deconvolution demonstration (Debluring)

Description

Here we try to deblur an image through a deconvolution problem. The convolution operator is the blur The problem can be expressed as this

Figure 3.11: Original image

This figure shows the original lena image.

$$arg\min_x \|Ax-b\|^2 + \tau\|H(x)\|_1$$

Where b is the degraded image, I the identity and A an operator representing the blur.
H is a linear operator projecting the signal in a sparse representation. Here we worked with wavelet.
Warning! Note that this demo require the LTFAT toolbox to work.
We set

- $f_1(x) = ||H(x)||_1$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|H(z)\|_1$$

- $f_2(x) = ||Ax-b||_2^2$ We define the gradient as:

$$\nabla_f(x) = 2A^*(Ax-b)$$

Results

References: [2]

3.2.6 DEMO_GRAPH_RECONSTRUCTION - Reconstruction of missing sample on a graph

Please see the GSPBOX for this demonstration. You can find it at:
http://lts2research.epfl.ch/gsp/
A demo of signal reconstruction is availlable at
https://lts2research.epfl.ch/gsp/doc/demos/gsp_demo_graph_tv.php

Figure 3.12: Depleted image
This figure shows the image after the application of the blur.

Figure 3.13: Reconstructed image
This figure shows the reconstructed image thanks to the algorithm.

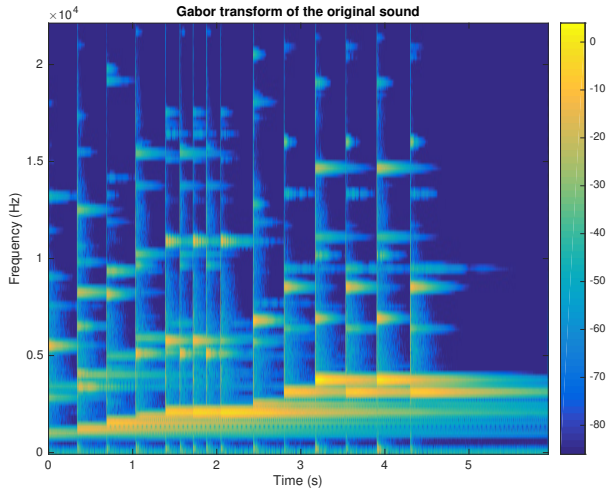


Figure 3.14: Original spectrogram

This figure shows the original spectrogram.

3.2.7 DEMO_SOUND_RECONSTRUCTION - Sound time in painting demonstration

Description

Here we solve a sound in-painting problem. The problem can be expressed as this

$$arg\min_x \|AG^*x - b\|^2 + \tau\|x\|_1$$

where b is the signal at the non clipped part, A an operator representing the mask selecting the non clipped part of the signal and G^* is the Gabor synthesis operation

Here the general assumption is that the signal is sparse in the Gabor domain! The noiseless particular case of this problem can be epressed as

$$arg\min_x \|x\|_1 \text{ s.t. } AG^*x = b$$

Warning! Note that this demo requires the LTFAT toolbox to work.

We set

- $f_1(x) = ||x||_1$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x - z\|_2^2 + \gamma\|z\|_1$$

- $f_2(x) = ||Ax - b||_2^2$ We define the gradient as:

$$\nabla_f(x) = 2 * GA^*(AG^*x - b)$$

Results

References: [3]

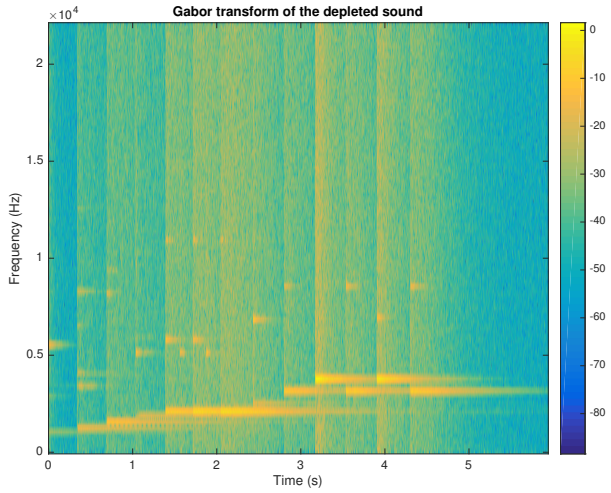


Figure 3.15: Spectrogram of the depleted sound

This figure shows the spectrogram after the loss of the sample (We loos 75% of the samples.)

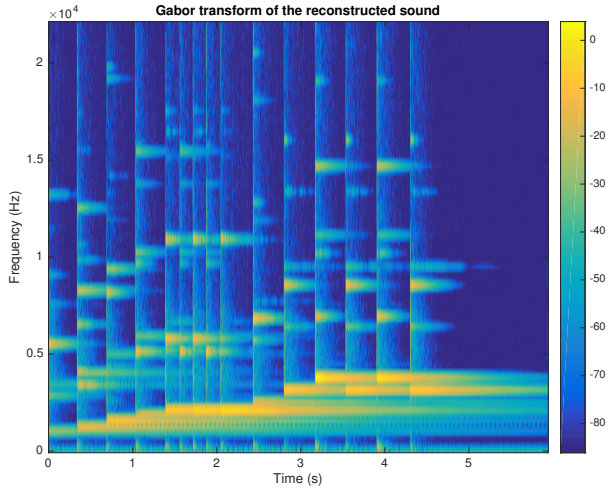


Figure 3.16: Spectrogram of the reconstructed sound

This figure shows the spectrogram of the reconstructed sound thanks to the algorithm.

3.2.8 DEMO_DOUGLAS_RACHFORD - Example of use of the douglas_rachford solver

Description

We present an example of the douglas_rachford solver through an image reconstruction problem. The problem can be expressed as this

$$arg\min_x \|x\|_{TV} \qquad such \quad that \qquad \|b-Ax\|_2 \leqslant \varepsilon$$

Where b is the degraded image, I the identity and A an operator representing the mask.

Note that the constraint can be inserted in the objective function thanks to the help of the indicative function. Then we recover the general formulation used for the solver of this toolbox.

We set

- $f_1(x) = ||x||_{TV}$ We define the prox of f_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_{TV}$$

- f_2 is the indicator function of the set S define by $||Ax-b||_2 < \varepsilon$ We define the prox of f_2 as

$$prox_{f2,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

with $i_S(x)$ is zero if x is in the set S and infinity otherwise. This previous problem has an identical solution as:

$$arg\min_z \|x-z\|_2^2 \qquad such \quad that \qquad \|Az-b\|_2 \leqslant \varepsilon$$

It is simply a projection on the B2-ball.

Results

References: [2]

3.2.9 DEMO_PIERRE - Demo to solve a particular l1 l2 problem

Description

The problem can be expressed like this

$$\arg\min_{c,b} \|s-\Psi c-\Phi b\|^2 + \mu_1\|c\|_1 + \mu_2\|b\|_1$$

Where s are the measurements, Ψ the Fourier matrix and $\Phi = \Phi * M$ with M a diagonal matrix with $+1,-1$ random values.

We will use generalized forward backward to solve this problem. The gradients of

$$\|s-\Psi c-\Phi b\|^2$$

are

$$\nabla_c f(c,b) = 2\Psi^*(\Psi c+\Phi b-s)$$
$$\nabla_b f(c,b) = 2\Phi^*(\Psi c+\Phi b-s)$$

In this code the variable b and c will be stack into one single vector of size $2N$

Figure 3.17: Original image

This figure shows the original Lena image.

Figure 3.18: Depleted image

This figure shows the image after the application of the mask. Note that 85% of the pixels have been removed.

Figure 3.19: Reconstructed image

This figure shows the reconstructed image thanks to the algorithm.

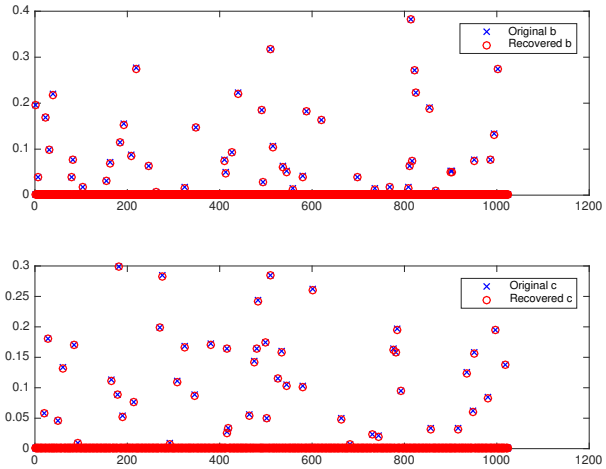


Figure 3.20: Results of the reconstruction

The support of the signal is recovered.

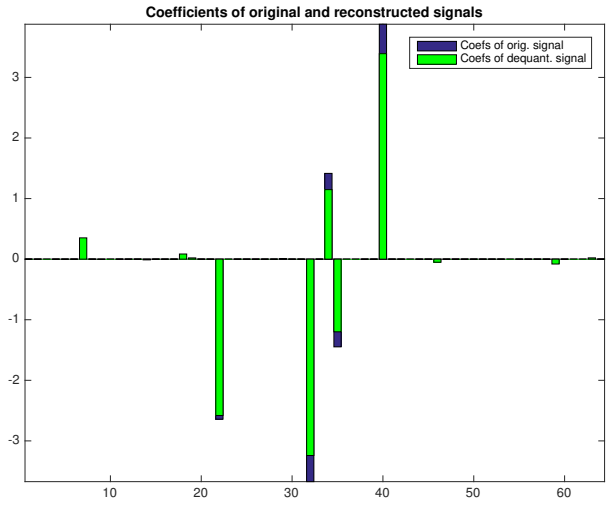


Figure 3.21: Original, quantized and dequantized signals

Results

3.2.10 DEMO_DEQUANTIZATION - Dequantization demo

Description

This demo shows how a quantized signal, sparse in the DCT domain, can be dequantized solving a convex problem using Douglas-Rachford algorithm

Suppose signal y has been quantized. In this demo we use quantization levels that are uniformly spread between the min. and max. value of the signal. The resulting signal is y_Q.

The problem can be expressed as

$$arg\min_x \|x\|_1 \text{ s.t. } \|Dx - y_Q\|_\infty \leqslant \frac{\alpha}{2}$$

where D is the synthesis dictionary (DCT in our case) and α is the distance between quantization levels. The constraint basically represents the fact that the reconstructed signal samples must stay within the corresponding quantization stripes.

After sparse coordinates are found, the dequantized signal is obtained simply by synthesis with the dictionary.

The program is solved using Douglas-Rachford algorithm. We set

- $f_1(x) = ||x||_1$. Its respective prox is the soft thresholding operator.
- $f_2(x) = i_C$ is the indicator function of the set C, defined as

$$C = \{x | \|Dx - y_Q\|_\infty <= \frac{\alpha}{2}\}$$

Its prox is the orthogonal projection onto that set, which is realized by entry-wise 1D projections onto the quantization stripes. This is realized for all the entries at once by function proj_box.

As an alternative, setting algorithm = 'LP' switches to computing the result via linear programming (requires Matlab optimization toolbox).

Results

References: [3]

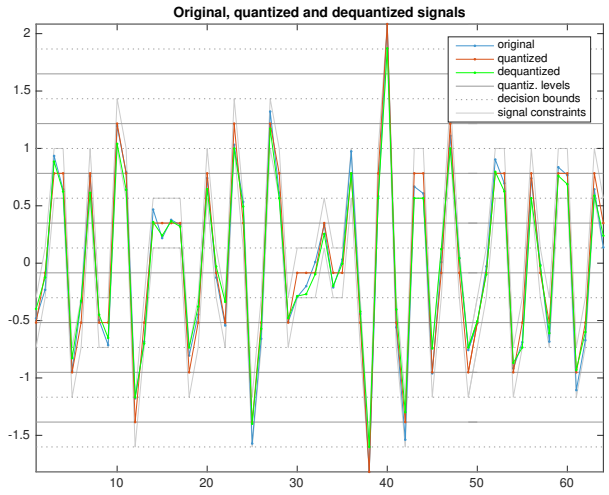


Figure 3.22: Quantization error and error of reconstruction (i.e. original - reconstr.)

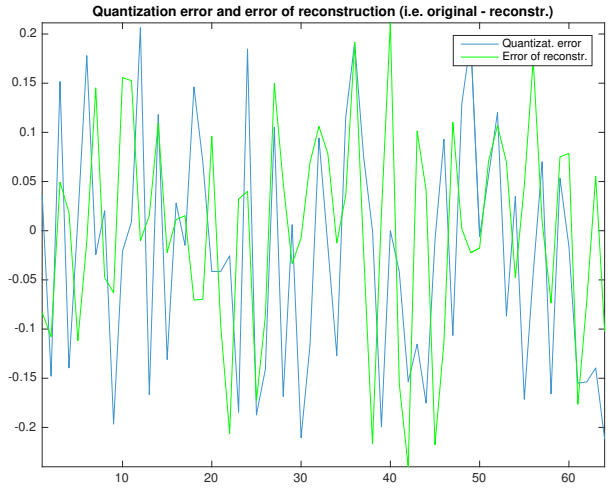


Figure 3.23: Coefficients of original and reconstructed signals

3.3 Other demo

3.3.1 DEMO_ADMM - Example of use of the ADMM solver

Description

The demo file present an example of the ADMM (alternating direction method of multipliers) solver. Unfortunately, this method is not fully automatic and the user needs to define the functions in a particular way.

Please read the paper of Boyd "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers" to be able to understand this demonstration file.

ADMM is used to solve problem of the form

$$sol = \min_x f_1(y) + f_2(x) \qquad s.t. \qquad y = Lx$$

In this demonstration file, we tackle the following problem

$$arg\min_x \tau \|Mx - z\|_2^2 + \|Lx\|_1$$

where z are the measurements, W the discrete wavelet transform, M a masking operator and τ a regularization parameter. Clearly, setting $Lx = y$ allows to recover the general form for ADMM problem. Contrarily to the other solvers of the UNLocBoX the solver require special proximal operators.

Here $f_1(x) = \tau\|Mx - z\|_2^2$ would normally take the following proximal operator:

```
f1.prox = @(x, t) ( 1 + tau * t * mask ).^(-1) .* ( x + tau * t * mask.*z);
f1.eval = @(x) tau * norm(mask .* x - z)^2;
```

which correspond to the solution of the following problem

$$prox_{f1,t}(z) = arg\min_x \frac{1}{2}\|x - z\|_2^2 + t\|Mx - y\|_2^2$$

However, the ADMM algorithm requires to solve a special proximal operator instead:

$$prox^L_{f1,t}(z) = arg\min_x \frac{1}{2}\|Lx - z\|_2^2 + t\|Mx - y\|_2^2$$

which is define in MATLAB as:

```
f1.proxL = @(x, t) ( 1 + tau * t * mask ).^(-1) .* ( Lt(x) + tau * t * mask.*z);
f1.prox = @(x, t) ( 1 + tau * t * mask ).^(-1) .* ( x + tau * t * mask.*z);
f1.eval = @(x) tau * norm(mask .* x - z)^2;
```

where *Lt* it the adjoint of the *L* (here the inverse wavelet transform) Because the wavelet transform is an orthonormal basis.

The function $f_2(y) = \|y\|_1$ is defined in MATLAB as:

```
param_l1.verbose = verbose - 1;
f2.prox = @(x, T) prox_l1(x, T, param_l1);
f2.eval = @(x) norm_l1(L(x));
f2.L = L;
f2.Lt = Lt;
```

Note the field *f2.L* and f2.Lt that indicate that the real function function is actually $f_2(Ly) = \|Lx\|_1$.

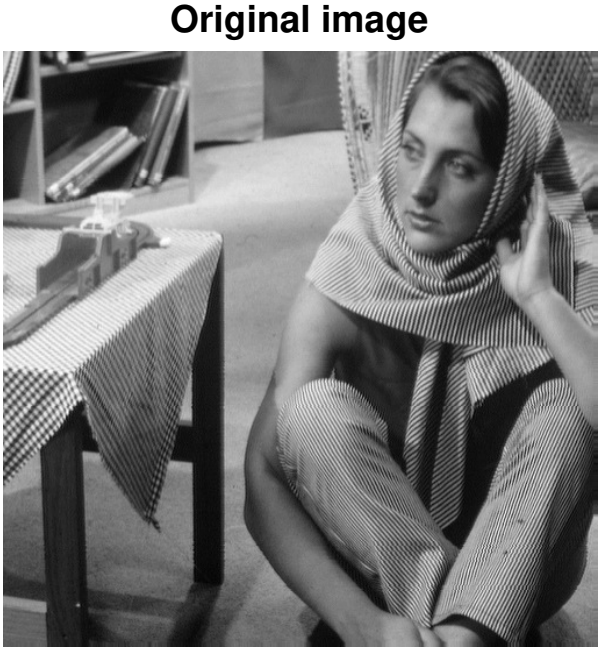


Figure 3.24: Original image

This figure shows the original Lena image.

Figure 3.25: Depleted image

This figure shows the image after the application of the mask and addition of the noise. Note that 50% of the pixels have been removed.

Figure 3.26: Reconstructed image

This figure shows the reconstructed image thanks to the algorithm.

Results

References: [2], [1]

3.3.2 DEMO_SDMM - Example of use of the sdmm solver

Description

We present an example of the solver through an image denoising problem. We express the problem cas

$$arg\min_x \|x-b\|_2^2 + \tau_1\|y\|_{TV} + \tau_2\|H(z)\|_1 \qquad such \quad that \qquad x = y = Hz$$

Where b is the degraded image, τ_1 and τ_2 two real positive constant and H a linear operator on x. H is a wavelet operator. We set:

- $g_1(x) = ||x||_{TV}$ We define the prox of g_1 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|z\|_{TV}$$

- $g_2(x) = ||H(x)||_1$ We define the prox of g_2 as:

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \|H(z)\|_1$$

- $f(x) = ||x-b||_2^2$ We define the gradient as:

$$\nabla_f(x) = 2(x-b)$$

Figure 3.27: Original image

This figure shows the original image (The cameraman).

Results

The rwt toolbox is needed to run this demo.

References: [2]

3.3.3 DEMO_WEIGHTED_L1 - Demonstration of the use of the bpdn solver

We solve a compress sensing problem in 2 dimensions.

$$arg\min_x \|\Psi x\|_1 s.t. \|y - Ax\|_2 < \varepsilon$$

We first solve the problem very generally. Then using the first solution, we define weight for the L1 norm and compute again the solution.

A is a mask operator in the Fourier domain. The measurements are done in the Fourier domain.

3.3.4 DEMO_TVDN - Demonstration of the use of the tvdn solver

In this demo we solve two different problems. Both can be written on this form:

$$arg\min_x \|x\|_{TV} s.t. \|y - Ax\|_2 < \varepsilon$$

The first problem is an inpainting problem with 33% of the pixel. In that case A is simply a mask and y the know pixels.

The second problem consists of reconstructing the image with only 33% of the Fourier coefficients. In that case A is a truncated Fourier operator.

Figure 3.28: Depleted image
This figure shows the image after addition of the noise

Figure 3.29: Reconstruted image
This figure shows the reconstructed image thanks to the algorithm.

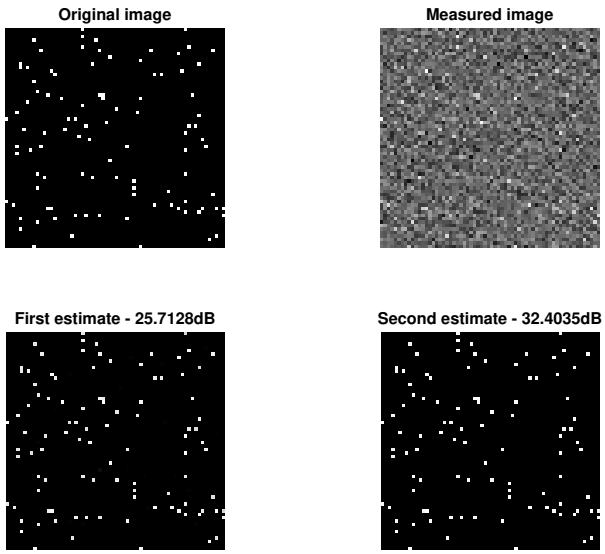


Figure 3.30: Figure

Results of the code

Figure 3.31: Original image

The cameraman

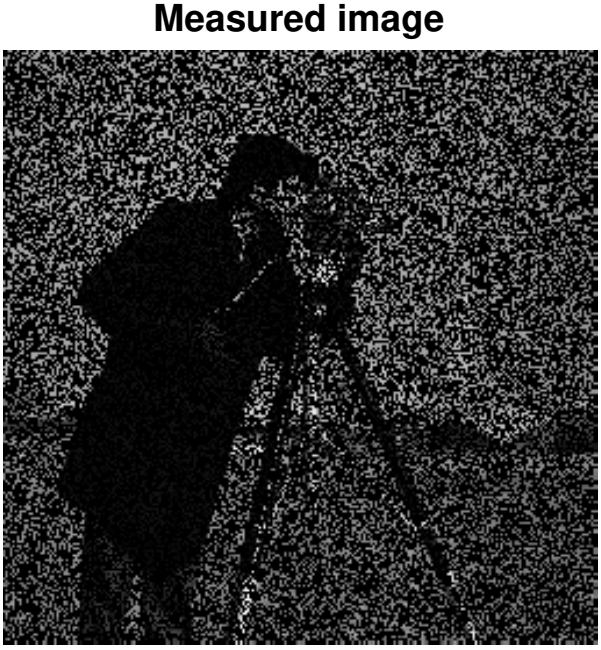


Figure 3.32: Measurements

Figure 3.33: In painting with 33% of known pixel and a SNR of 30dB

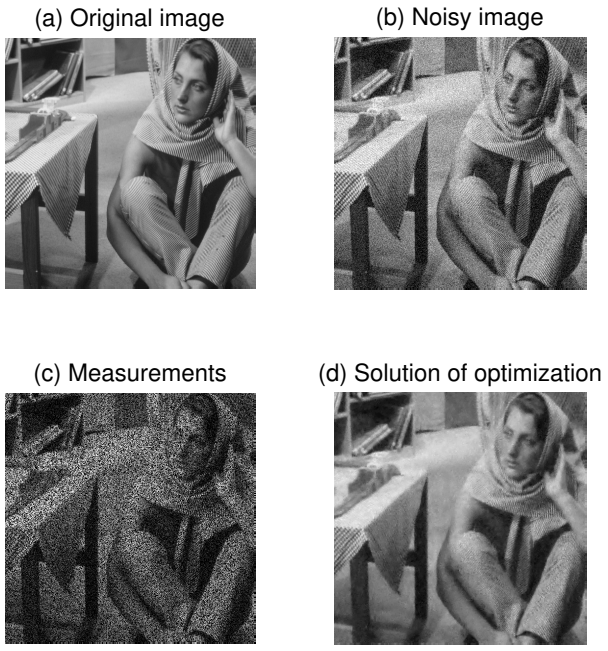


Figure 3.34: Results

3.3.5 DEMO_FBB_PRIMAL_DUAL - Example of use of the forward backward based primal dual solver

Description

We present an example of the the forward backward based primal dual solver through an image de-noising, in-painting problem. We express the problem in the following way

$$arg\min_x \|A(x-b)\|^2 + \lambda\|x\|_{TV} + \tau\|Wx\|_1$$

Where b is the degraded image, W the wavelet transform and A a linear operator performing the masking operation. This operator set to 0 all unknown pixels.

Results

References: [6]

Chapter 4

Unlocbox - Utils

4.1 Norms

4.1.1 NORM_TV - 2 Dimentional TV norm

Usage

```
y = norm_tv(x);
y = norm_tv(I,wx,wy);
```

Input parameters

I	Input data
wx	Weights along x
wy	Weights along y

Output parameters

y	Norm

Description

Compute the 2-dimentional TV norm of I. If the input I is a cube. This function will compute the norm of all image and return a vector of norms.

4.1.2 NORM_TV1D - 1 Dimentional TV norm

Usage

```
y = norm_tv1d(x)
y = norm_tv1d(x,w)
```

Input parameters

I	Input data
w	Weights

Output parameters

y	Norm

Description

Compute the 1-dimentional TV norm of I. If the input I is a matrix. This function will compute the norm of all line and return a vector of norms.

4.1.3 NORM_TV3D - 3 Dimentional TV norm

Usage

```
y = norm_tv3d(x)
y = norm_tv3d(x, wx, wy, wz )
```

Input parameters

x	Input data (3 dimentional matrix)
wx	Weights along x
wy	Weights along y
wz	Weights along z

Output parameters

y	Norm

Description

Compute the 3-dimentional TV norm of x. If the input I is a 4 dimentional signal. This function will compute the norm of all cubes and return a vector of norms.

4.1.4 NORM_TV4D - 4 Dimentional TV norm

Usage

```
y = norm_tv4d(x)
y = norm_tv4d(x, wx, wy, wz, wt )
```

Input parameters

x	Input data (3 dimentional matrix)
wx	Weights along x
wy	Weights along y
wz	Weights along z
wt	Weights along t

Output parameters

y	Norm

Description

Compute the 4-dimentional TV norm of x. If the input I is a 5 dimentional signal. This function will compute the norm of all 4 dimentional cubes and return a vector of norms.

4.1.5 NORM_TVND - N Dimentional TV norm

Usage

```
norm_tvnd(x,weights)
```

Input parameters

x	Input data (N dimentional matrix)
type	type ('isotropic' or 'anisotropic') (default 'isotropic')
weights	Weights

Output parameters

sol	Norm

Description

Compute the N-dimentional TV norm of x

4.1.6 NORM_L21 - L21 mixed norm

Usage

```
n21 = norm_l21(x);
n21 = norm_l21(x, g_d,g_t);
n21 = norm_l21(x, g_d,g_t, w2,w1);
```

Input parameters

x	Input data
g_d	group vector 1
g_t	group vector 2
w2	weights for the two norm (default 1)
w1	weights for the one norm (default 1)

Output parameters

y	Norm

Description

`norm_l21(x, g_d,g_t, w2,w1)` returns the norm L21 of x. If x is a matrix the 2 norm will be computed as follow:

$$\|x\|_{21} = \sum_j \left| \sum_i |x(i,j)|^2 \right|^{1/2}$$

In this case, all other argument are not necessary.

'norm_l21(x)' with x a row vector is equivalent to norm(x,1) and 'norm_l21(x)' with x a line vector is equivalent to norm(x)

For fancy group, please provide the groups vectors.

`g_d`, `g_t` are the group vectors. `g_d` contain the indices of the element to be group and `g_t` the size of different groups.

Example: x=[x1 x2 x3 x4 x5 x6] Group 1: [x1 x2 x4 x5] Group 2: [x3 x6]

Leads to

=> g_d=[1 2 4 5 3 6] and g_t=[4 2] Or this is also possible => g_d=[4 5 3 6 1 2] and g_t=[2 4]

This function works also for overlapping groups.

4.1.7 NORM_Linf1 - Linf1 mixed norm

Usage

```
ninf1 = norm_linf1(x);
ninf1 = norm_linf1(x, g_d,g_t);
ninf1 = norm_linf1(x, g_d,g_t, winf,w1);
```

Input parameters

x	Input data
g_d	group vector 1
g_t	group vector 2
winf	weights for the sup norm (default 1)
w1	weights for the one norm (default 1)

Output parameters

y	Norm

Description

`norm_linf1(x, g_d,g_t, w2,w1)` returns the norm Linf1 of x. If x is a matrix the sup norm will be computed over the lines (2nd dimention) and the one norm will be computed over the rows (1st dimention). In this case, all other argument are not necessary.

$$\|x\|_{\infty 1} = \sum_j \left| \max_i |x(i,j)| \right|$$

'norm_linf1(x)' with x a row vector is equivalent to norm(x,1) and 'norm_linf1(x)' with x a line vector is equivalent to max(abs(x))

For fancy group, please provide the groups vectors.

`g_d`, `g_t` are the group vectors. `g_d` contain the indices of the element to be group and `g_t` the size of different groups.

Example: x=[x1 x2 x3 x4 x5 x6] Group 1: [x1 x2 x4 x5] Group 2: [x3 x6]

Leads to

=> g_d=[1 2 4 5 3 6] and g_t=[4 2] Or this is also possible => g_d=[4 5 3 6 1 2] and g_t=[2 4]

This function works also for overlapping groups.

4.1.8 NORM_NUCLEAR - - Nuclear norm of x

Usage

```
norm_nuclear(x)
```

Input parameters

x	a matrix

Output parameters

n	nuclear norm of x

4.1.9 NORM_SUMG - 2 Dimentional TV norm

Usage

```
y = norm_sumg(x, G);
y = norm_sumg(x, G, w);
```

Input parameters

x	Input data (vector)
G	The structure array of norm operator:
w	Weights (default 1)

Output parameters

n	Norm

Description

`n = norm_sumg(x, G, w)` returns the sum of the norm x given in the structure array G. The norm can be weighted using the parameter `weights`.

4.2 Operators

4.2.1 GRADIENT_OP - 2 Dimensional gradient operator

Usage

```
[dx, dy] = gradient_op(I)
[dx, dy] = gradient_op(I, wx, wy)
```

Input parameters

I	Input data
wx	Weights along x
wy	Weights along y

Output parameters

dx	Gradient along x
dy	Gradient along y

Description

Compute the 2-dimensional gradient of I. If the input I is a cube. This function will compute the gradient of all image and return two cubes.

4.2.2 GRADIENT_OP3D - 3 Dimentional gradient operator

Usage

```
[dx, dy, dz] = gradient_op3d(I)
[dx, dy, dz] = gradient_op3d(I, wx, wy, wz)
```

Input parameters

I	Input data
wx	Weights along x
wy	Weights along y
wz	Weights along z

Output parameters

dx	Gradient along x
dy	Gradient along y
dz	Gradient along z

Description

Compute the 3-dimentional gradient of I. If the input I has 4 dimentions. This function will compute the gradient of all cubes and return 3 4-dimentionals signals

4.2.3 GRADIENT_OP4D - 4 Dimentional gradient operator

Usage

```
[dx, dy, dz, dt] = gradient_op4d(I)
[dx, dy, dz, dt] = gradient_op4d(I, wx, wy, wz, wt)
```

Input parameters

I	Input data
wx	Weights along x
wy	Weights along y
wz	Weights along z
wt	Weights along t

Output parameters

dx	Gradient along x
dy	Gradient along y
dz	Gradient along z
dt	Gradient along t

Description

Compute the 4-dimentional gradient of I. If the input I has 5 dimentions. This function will compute the gradient of all 4 dimentional cubes and return 4 5-dimentionals signals

4.2.4 GRADIENT_OP1D - 1 Dimentional gradient operator

Usage

```
dx = gradient_op1d(I)
dx = gradient_op1d(I, wx)
```

Input parameters

I	Input data
wx	Weights along x

Output parameters

dx	Gradient along x

Description

Compute the 1-dimentional gradient of I. If the input I is a matrix. This function will compute the gradient of all vectors and return a matrix.

4.2.5 DIV_OP - Divergence operator in 2 dimensions

Usage

```
I = div_op(dx, dy)
I = div_op(dx, dy, wx, wy)
```

Input parameters

dx	Gradient along x
dy	Gradient along y
wx	Weights along x
wy	Weights along y

Output parameters

I	Output divergence image

Description

Compute the 2-dimensional divergence of an image. If a cube is given, it will compute the divergence of all images in the cube.

Warning: computes the divergence operator defined as minus the adjoint of the gradient

$$\text{div} = -\nabla^*$$

4.2.6 DIV_OP3D - Divergence operator in 3 dimentions

Usage

```
I = div_op3d(dx, dy, dz)
I = div_op3d(dx, dy, dz, wx, wy, wz)
```

Input parameters

dx	Gradient along x
dy	Gradient along y
dz	Gradient along z
wx	Weights along x
wy	Weights along y
wz	Weights along z

Output parameters

I	Output image

Description

Compute the 3-dimentional divergence of a 3D-image. If a 4 dimentional signal is given, it will compute the divergence of all cubes in the 4 diementionals signal.

Warning this function compute the divergence operator defined as minus the adjoint of the gradient

$$\text{div} = -\nabla^*$$

4.2.7 DIV_OP4D - Divergence operator in 4 dimentions

Usage

```
I = div_op4d(dx, dy, dz, dt)
I = div_op4d(dx, dy, dz, dt, wx, wy, wz, wt)
```

Input parameters

dx	Gradient along x
dy	Gradient along y
dz	Gradient along z
dt	Gradient along t
wx	Weights along x
wy	Weights along y
wz	Weights along z
wt	Weights along t

Output parameters

I	Output image

Description

Compute the 4-dimentional divergence of a 4D-image. If a 5 dimentional signal is given, it will compute the divergence of all 4 dimentional cubes in the 5 diementionals signal.

Warning this function compute the divergence operator defined as minus the adjoint of the gradient

$$\text{div} = -\nabla^*$$

4.2.8 DIV_OP1D - Divergence operator in 1 dimention

Usage

```
I = div_op1d(dx)
I = div_op1d(dx, wx)
```

Input parameters

dx	Gradient along x
wx	Weights along x

Output parameters

I	Output divergence vector

Description

Compute the 1-dimentional divergence of a vector. If a matrix is given, it will compute the divergence of all vectors in the matrix.

Warning this function compute the divergence operator defined as minus the adjoint of the gradient

$$\text{div} = -\nabla^*$$

4.2.9 LAPLACIAN_OP - 2 dimentional Laplacian

Usage

```
[I] = laplacian_op( I );
```

Input parameters

I	Input image

Output parameters

I	Laplacian

Description

Compute the sum of the laplacian along x and y. This operator is self-adjoint.

$$\mathscr{L} = I_{xx} + I_{yy}$$

4.2.10 LAPLACIANX_OP - dimentional Laplacian

Usage

```
[Lx] = laplacianx_op( I );
```

Input parameters

I	Input image

Output parameters

Lx	Laplacian along x

Description

Compute the sum of the laplacian along x. This operator is self-adjoint.

$$\mathscr{L}_x = I_{xx}$$

4.2.11 LAPLACIANY_OP - dimentional Laplacian

Usage

```
[Ly] = laplaciany_op( I );
```

Input parameters

I	Input image

Output parameters

Ly	Laplacian along y

Description

Compute the sum of the laplacian along y. This operator is self-adjoint.

$$\mathscr{L}_y = I_{yy}$$

4.3 Other

4.3.1 SNR - Compute the SNR between two maps

Usage

```
snr_val = snr(map_init, map_noisy)
```

Input parameters

map_init	initial signal
map_recon	noisy signal

Output parameters

snr_val	snr

Description

computes the SNR between the maps `map_init` and `map_noisy`. The SNR is computed as:

```
10 * log10( var(map_init) / var(map_init-map_noisy) )
```

where var stands for the matlab built-in function that computes the variance.

4.3.2 SOFT_THRESHOLD - soft thresholding

Usage

```
sz = soft_threshold(z,T);
```

Input parameters

z	Input signal
T	Threshold if T is a vector, then thresholding is applied component-wise

Output parameters

sz	Soft thresholded signal

Description

This function soft thresholds z by T. It can handle complex input z.

4.3.3 SET_SEED - sets the seed of the default random random generator

Usage

```
set_seed(my_seed)
set_seed()
```

Input parameters

my_seed	new_seed

Description

Set the seed of the default random random generator

4.3.4 VEC - vectorize x

Usage

```
r = vec(x);
```

Description

Inputs parameters: x : vector or matrix

Outputs parameters: r : row vector

This function vectorize x.

4.3.5 SVDECON - Fast svds when n«m

Usage

```
[U,S,V] = svdecon(X);
```

Input parameters

X	Input data (n x m)

Output parameters

U	Left singular vectors
S	Singular values
U	Right signular vectors

Description

This function is an acceleration of svd. It is particularly efficient when n«m

4.3.6 SVDSECON - Fast svds when n«m

Usage

```
[U,S,V] = svdsecon(X,k);
```

Input parameters

X	Input data (n x m)
k	Number of singular values

Output parameters

U	Left singular vectors
S	Singular values
U	Right signular vectors

Description

This function is an acceleration of svds. It is particularly efficient when n«m

4.3.7 SUM_SQUAREFORM - sparse matrix that sums the squareform of a vector

Usage

```
[S, St] = sum_squareform(n)
[S, St] = sum_squareform(n, mask)
```

Input parameters

n	size of matrix W
mask	if given, S only contain the columns indicated by the mask

Output parameters

S	matrix so that S*w = sum(W) for vector w = squareform(W)
St	the adjoint of S

Description

Creates sparse matrices S, St = S' so that S*w = sum(W), where w = squareform(W)

The mask is used for large scale computations where only a few non-zeros in W are to be summed. It needs to be the same size as w, n(n-1)/2 elements. See the example below for more details of usage.

Properties of S: * size(S) = [n, (n(n-1)/2)] % if no mask is given. * size(S, 2) = nnz(w) % if mask is given * norm(S)^2 = 2(n-1) * sum(S) = 2*ones(1, n*(n-1)/2) * sum(St) = sum(squareform(mask)) -- for full mask = (n-1)*ones(n,1)

Example:: % if mask is given, the resulting S are the ones we would get with the % following operations (but memory efficiently): [S, St] = sum_squareform(n); [ind_i, ~, w] = find(mask(:)); % get rid of the columns of S corresponding to zeros in the mask S = S(:, ind_i); St = St(ind_i, :);

4.3.8 SQUAREFORM_SP - Sparse counterpart of matlab's squareform

Usage

```
w = squareform_sp(W);
```

Input parameters

w	sparse vector with n(n-1)/2 elements OR
W	matrix with size [n, n] and zero diagonal

Output parameters

W	matrix form of input vector w OR
w	vector form of input matrix W

Description

This function is to be used instead of squareform.m when the matrix W or the vector w is sparse. For large scale computations, e.g. for learning the graph structure of a big graph it is necessary to take into account the sparsity.
Example:

```
B = sprand(8, 8, 0.1);
B = B+B';
B(1:9:end) = 0;
b = squareform_sp(B);
Bs = squareform_sp(b);
```

4.3.9 ZERO_DIAG - sets the diagonal of a matrix to 0

Usage

```
B = zero_diag(A);
```

Input parameters

A	input matrix

Output parameters

B	output with zero diagonal

Description

Works also for non-square matrices

Chapter 5

UNLocBoX - Signals

5.1 Tutorial demos

5.1.1 BARBARA - Load the 'barbara' test signal

Description

`barbara` loads the 'barbara' signal. Barbara is an image commonly used in image compression and filtering papers because it contains a range of tones and many thin line patterns. The resolution is (512 x 512).

This signal, and other standard image tests signals, can be found on Morgan McGuire's Computer Graphics Archive‘<http://graphics.cs.williams.edu/data/images.xml>‘_.

For convenience the output image is normalized by 255 and converted to double.

Example

Load the image and display it:

```
im = barbara();
imagescgray(im);
```

5.1.2 MANDRILL - Load the 'mandrill' test signal

Usage

```
im = mandrill();
im = mandrill(color);
```

Input parameters

color	boolean

Output parameters

none

Description

`mandrill()` loads the graylevel 'peppers' signal. Peppers is a common image processing test image of resolution (512 x 512).

`mandrill(1)` loads the color 'peppers' signal.

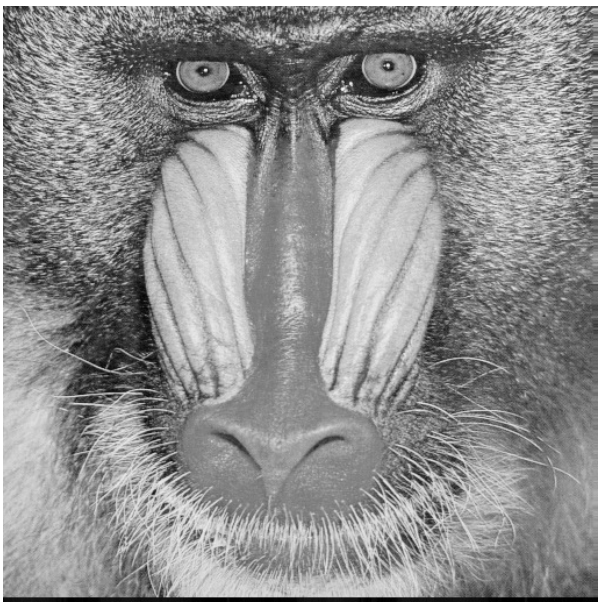
This signal, and other standard image tests signals, can be found on Morgan McGuire's Computer Graphics Archive‘<http://graphics.cs.williams.edu/data/images.xml>‘_.

For convenience the output image is normalized by 255 and converted to double.

Example

Load the image and display it:

```
im = mandrill();
imagescgray(im);
```



5.1.3 CAMERAMAN - Load the 'cameraman' test signal

Description

`cameraman` loads the 'cameraman' signal. The Cameraman (a.k.a. Photographer) is an image commonly used in image processing, especially filtering papers. The resolution is (256 x 256).

This signal, and other standard image tests signals, can be found on Morgan McGuire's Computer Graphics Archive‘<http://graphics.cs.williams.edu/data/images.xml>‘_.

For convenience the output image is normalized by 255 and converted to double.

Example

Load the image and display it:

```
im = cameraman();
imagescgray(im);
```

5.1.4 PEPPERS - Load the 'peppers' test signal

Usage

```
im = peppers();
im = peppers(color);
```

Input parameters

color	boolean

Output parameters

none

Description

`peppers()` loads the graylevel 'peppers' signal. Peppers is a common image processing test image of resolution (512 x 512).

`peppers(1)` loads the color 'peppers' signal.

This signal, and other standard image tests signals, can be found on Morgan McGuire's Computer Graphics Archive'<http://graphics.cs.williams.edu/data/images.xml>'_.

For convenience the output image is normalized by 255 and converted to double.

Example

Load the image and display it:

```
im = peppers();
imagescgray(im);
```

5.1.5 CHECKERBOARD - Load the 'checkerboard' test signal

Usage

```
im = checkerboard();
```

Input parameters

non none

Output parameters

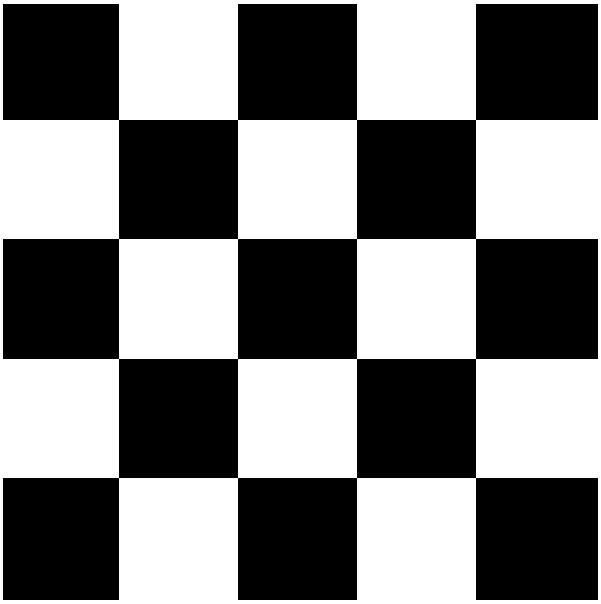
im image

Description

Example

Load the image and display it:

```
im = checkerboard();
imagescgray(im);
```



Bibliography

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[2] P. Combettes and J. Pesquet, "Proximal splitting methods in signal processing," *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pp. 185–212, 2011.

[3] ——, "A douglas–rachford splitting approach to nonsmooth convex variational signal recovery," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 1, no. 4, pp. 564–574, 2007.

[4] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[5] H. Raguet, J. Fadili, and G. Peyré, "Generalized forward-backward splitting," *arXiv preprint arXiv:1108.4404*, 2011.

[6] N. Komodakis and J.-C. Pesquet, "Playing with duality: An overview of recent primal-dual approaches for solving large-scale optimization problems," *arXiv preprint arXiv:1406.5429*, 2014.

[7] M. Fadili and J. Starck, "Monotone operator splitting for optimization problems in sparse recovery," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 2009, pp. 1461–1464.

[8] E. Van Den Berg and M. P. Friedlander, "Probing the pareto frontier for basis pursuit solutions," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 890–912, 2008.

[9] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *arXiv preprint arXiv:1108.0775*, 2011.

[10] M. Kowalski, K. Siedenburg, and M. Dorfler, "Social sparsity! neighborhood systems enrich structured shrinkage operators," *Signal Processing, IEEE Transactions on*, vol. 61, no. 10, pp. 2498–2511, 2013.

[11] M. Kowalski, "Sparse regression using mixed norms," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 303–324, 2009.

[12] M. Kowalski and B. Torresani, "Sparsity and persistence: mixed norms provide simple signal models with dependent coefficients," *Signal, image and video processing*, vol. 3, no. 3, pp. 251–264, 2009.

[13] A. Beck and M. Teboulle, "Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems," *Image Processing, IEEE Transactions on*, vol. 18, no. 11, pp. 2419–2434, 2009.

[14] L. Condat, "A direct algorithm for 1d total variation denoising," *IEEE Signal Processing Letters*, vol. 20, no. 11, pp. 1054–1057, 2013.

[15] N. Perraudin, D. Shuman, G. Puy, and P. Vandergheynst, "UNLocBoX A matlab convex optimization toolbox using proximal splitting methods," *ArXiv e-prints*, Feb. 2014.